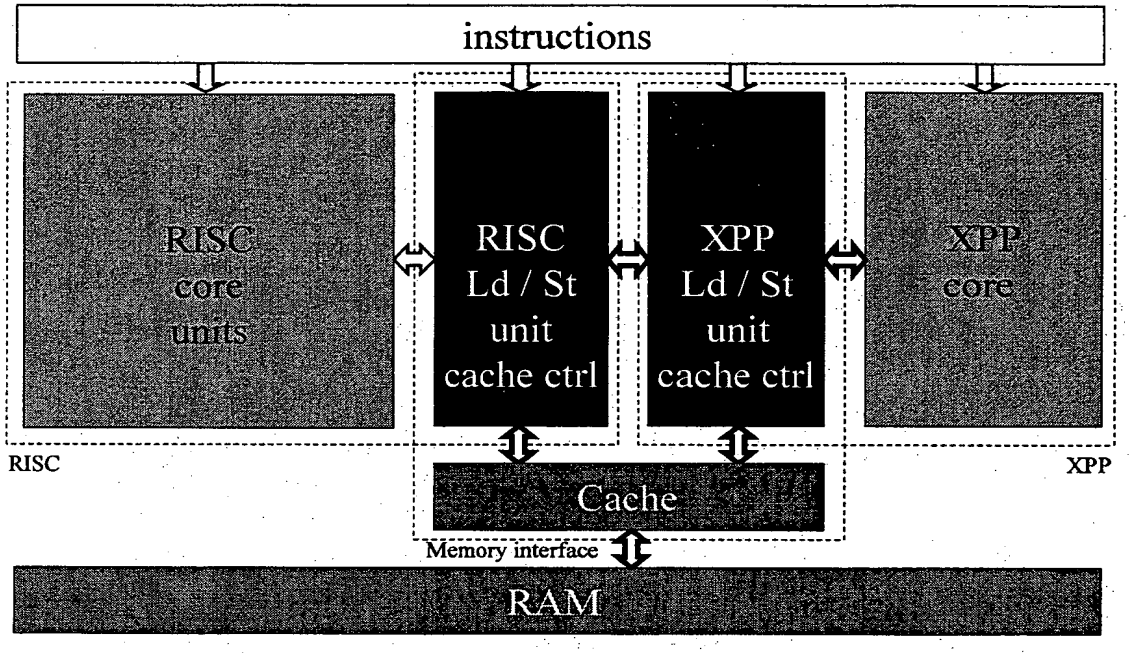
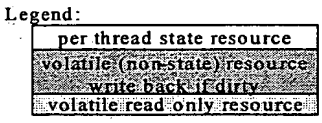


Fig. 1: Memory interface

```
XPPPreloadConfig( __XppCfg_foo );
for (int i=0; i <1000; ++i) {
 XPPPreload( 2, &a[i*30], 30 );
 XPPPreload( 0, &b[i*200], 200 );
 XPPPreloadClean( 5, &c[i*10], 10 );
 XPPExecute( );
/*
 Other RISC computations ...
 In the meanwhile the burst preloads and
 the previous configuration are running;
 The new configuration is executed as soon
 as the preloads and the previous
 configuration are finished.
 New burst preloads can be issued
 according to the FIFO length.
*/
}
Note: in all places where constants are used,
the value should actually come from a register
```



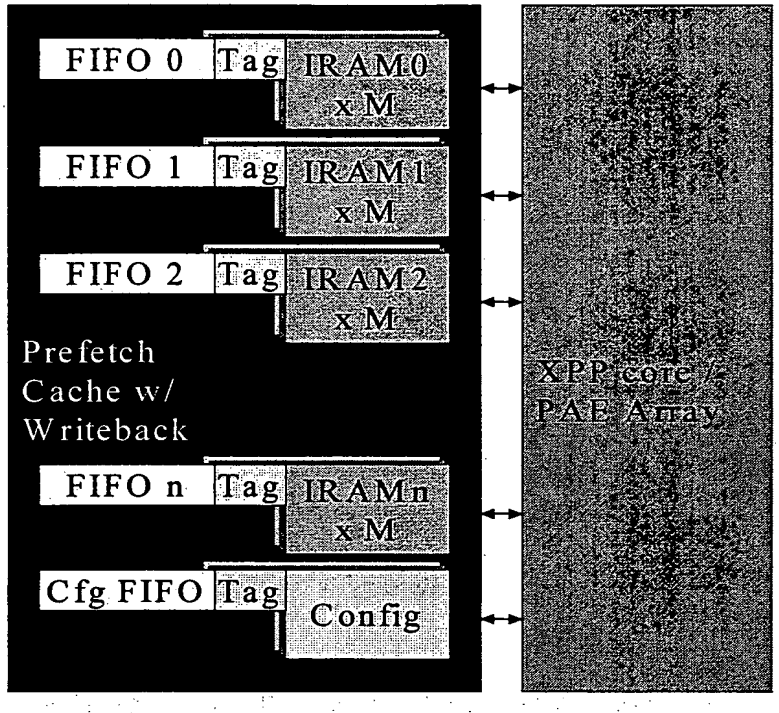


Fig. 2: IRAM & configuration cache controller data structures and usage example

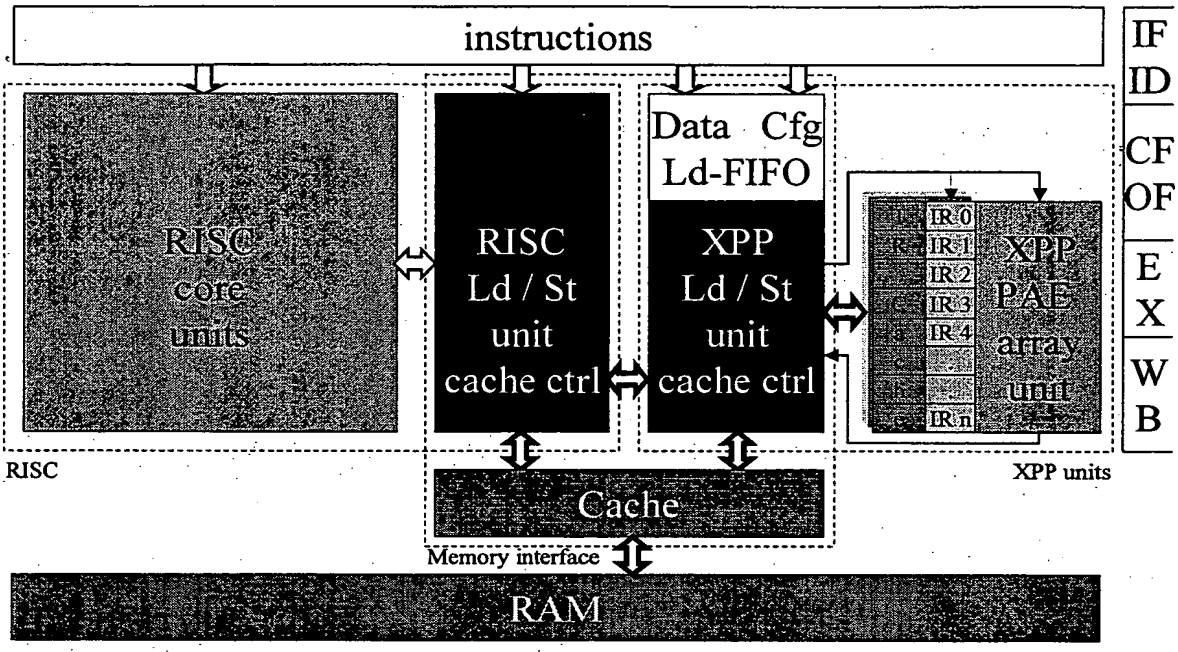


Fig. 3: Asynchronous pipeline of the XPP

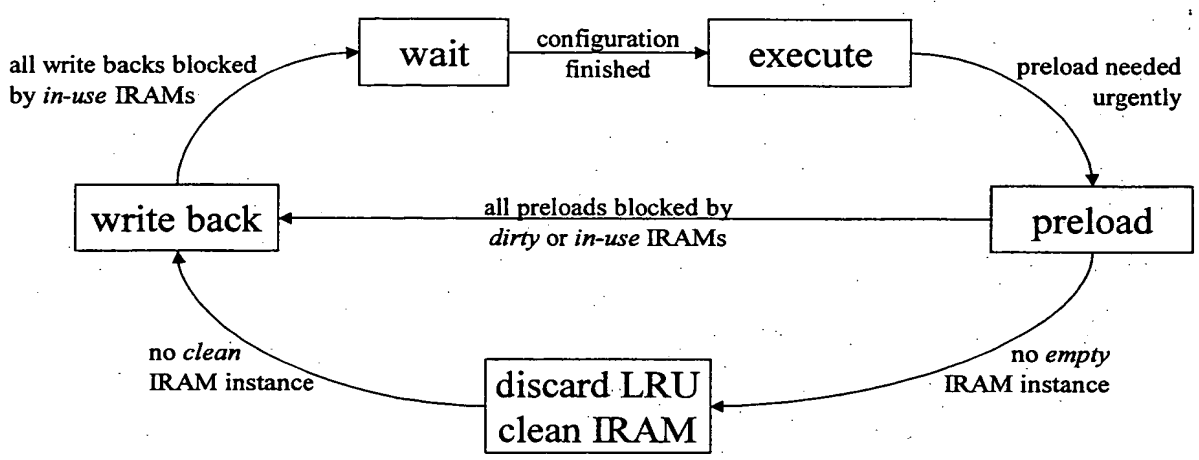


Fig. 4: State transition diagram for the XPP cache controller

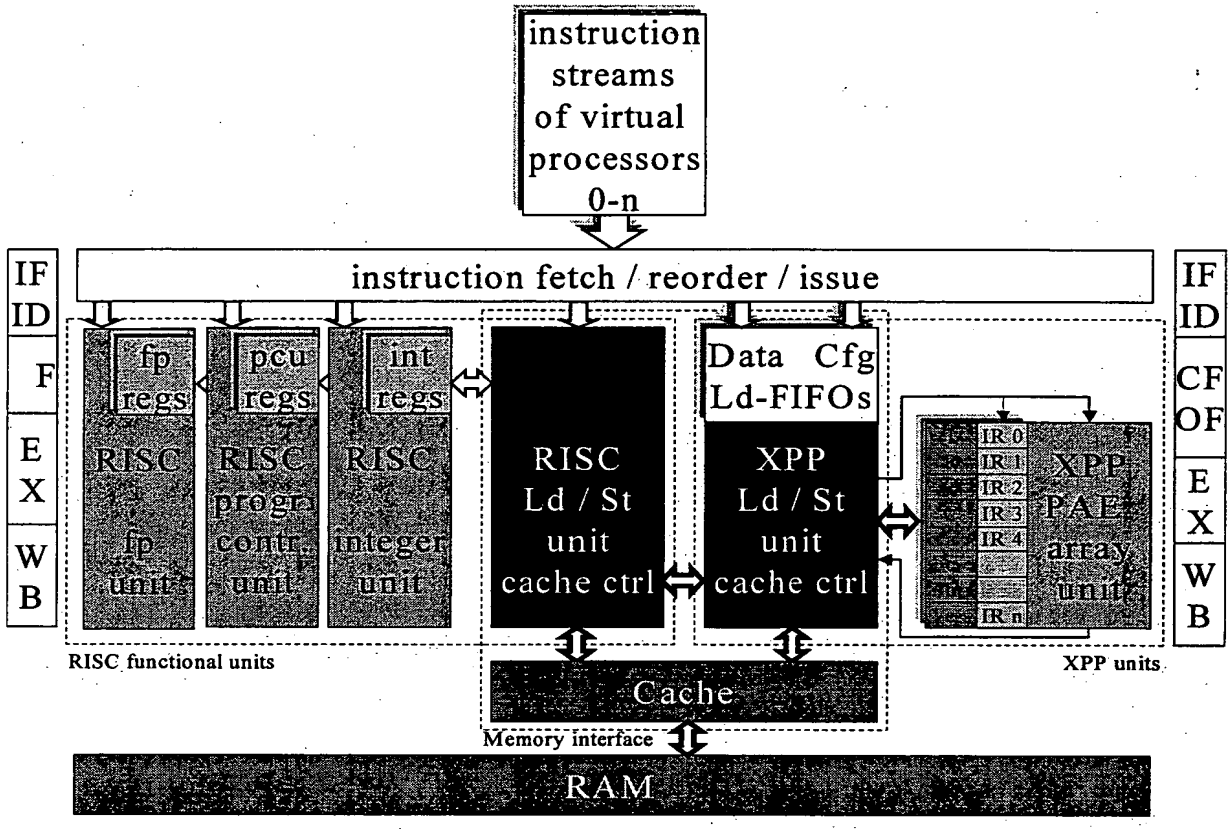


Fig. 5: Adding simultaneous multithreading

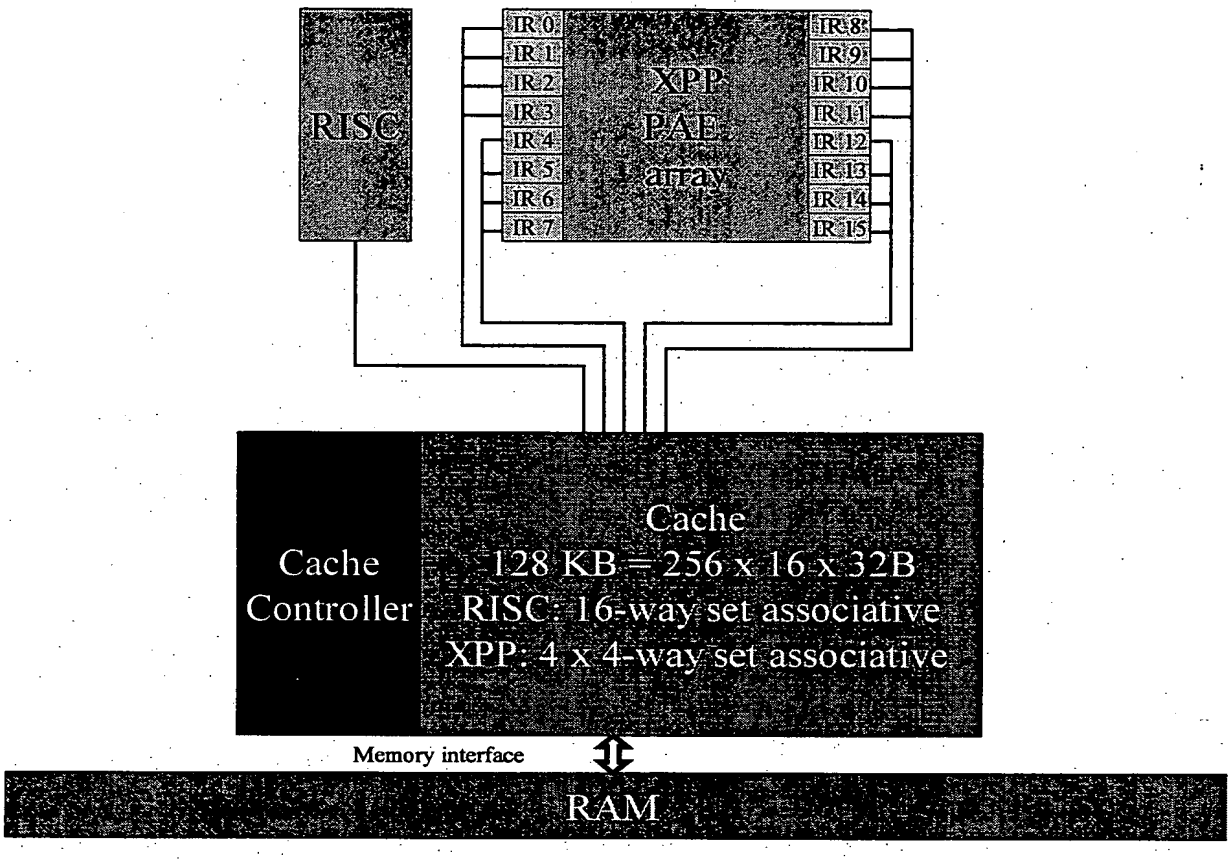


Fig. 6: Cache structure example

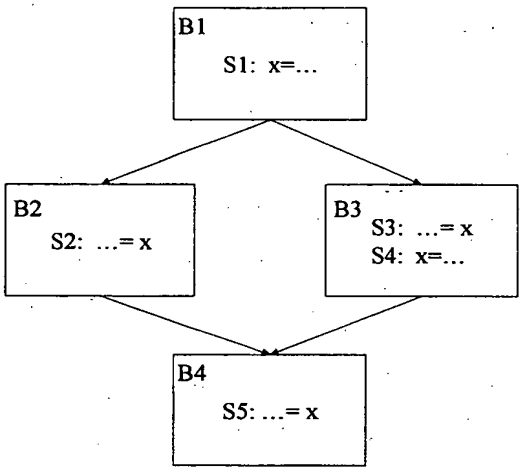


Fig. 7:Control-flow graph of a piece of program

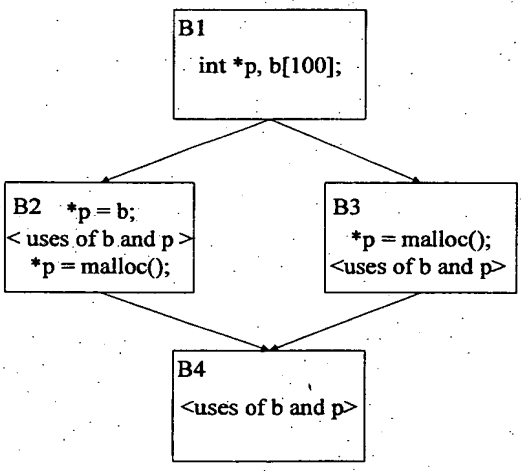


Fig. 8: Example of control-flow sensitivity

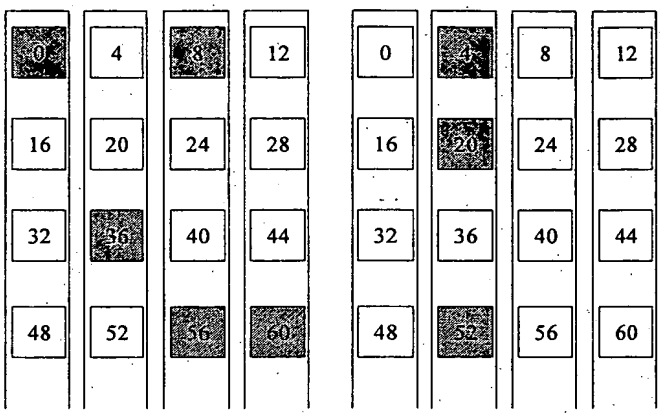


Fig. 9

```
for(j=1;j<=N-1;i++)
   for(j=1;j<=N;j++)
      b[i][j] = 0.25*(a[i-1][j] + a[i][j-1] +
a[i+1][j] + a[i][j+1]);
```

Fig. 10: Example for array merging

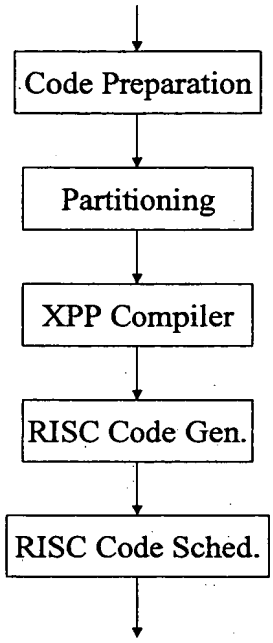


Fig. 11: Global View of the Compiling Process

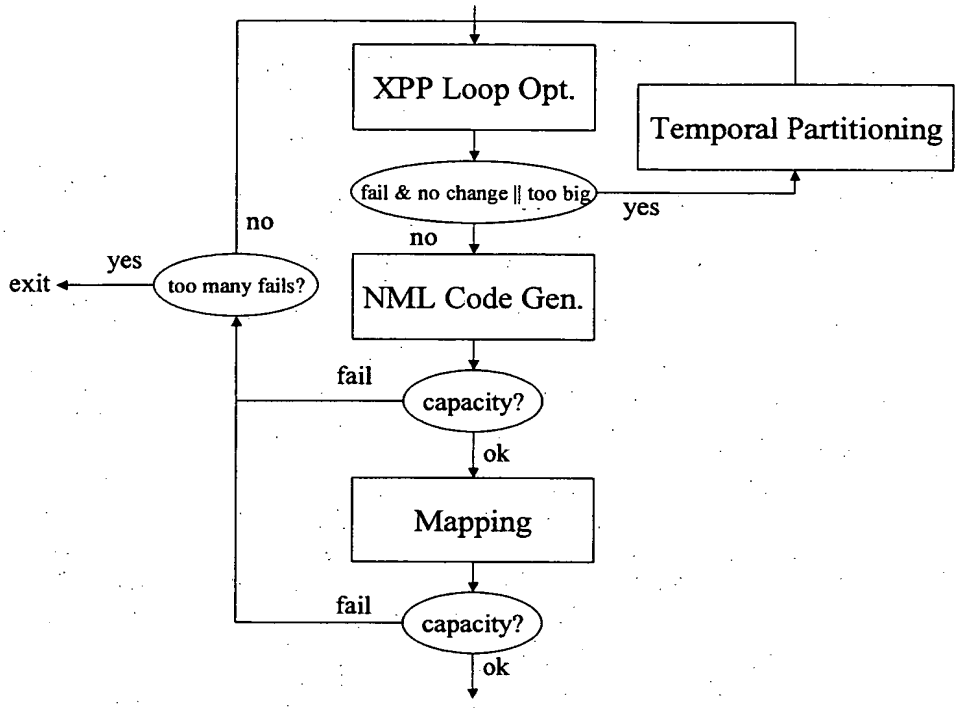


Fig. 12: Detailed Architecture of the XPP Compiler

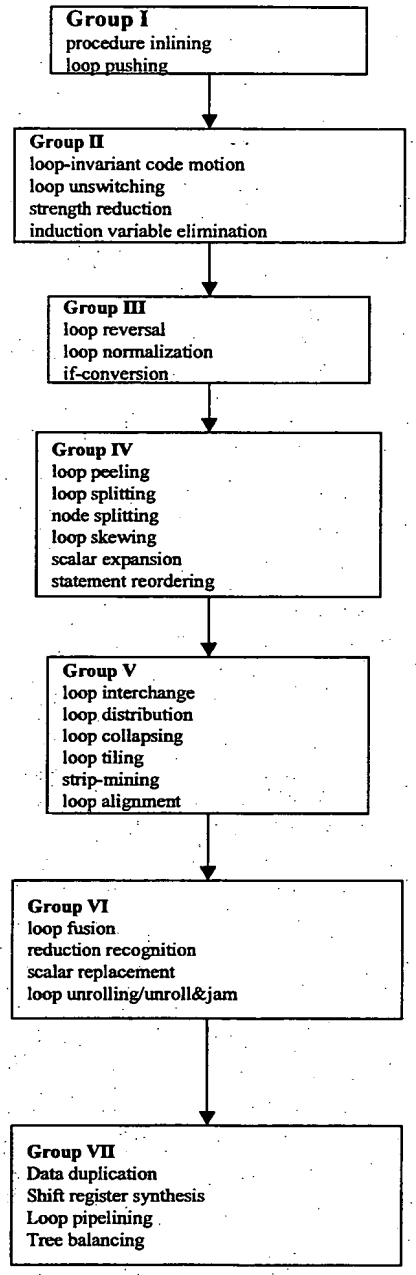


Fig. 13: Detailed View of the XPP Loop Optimization

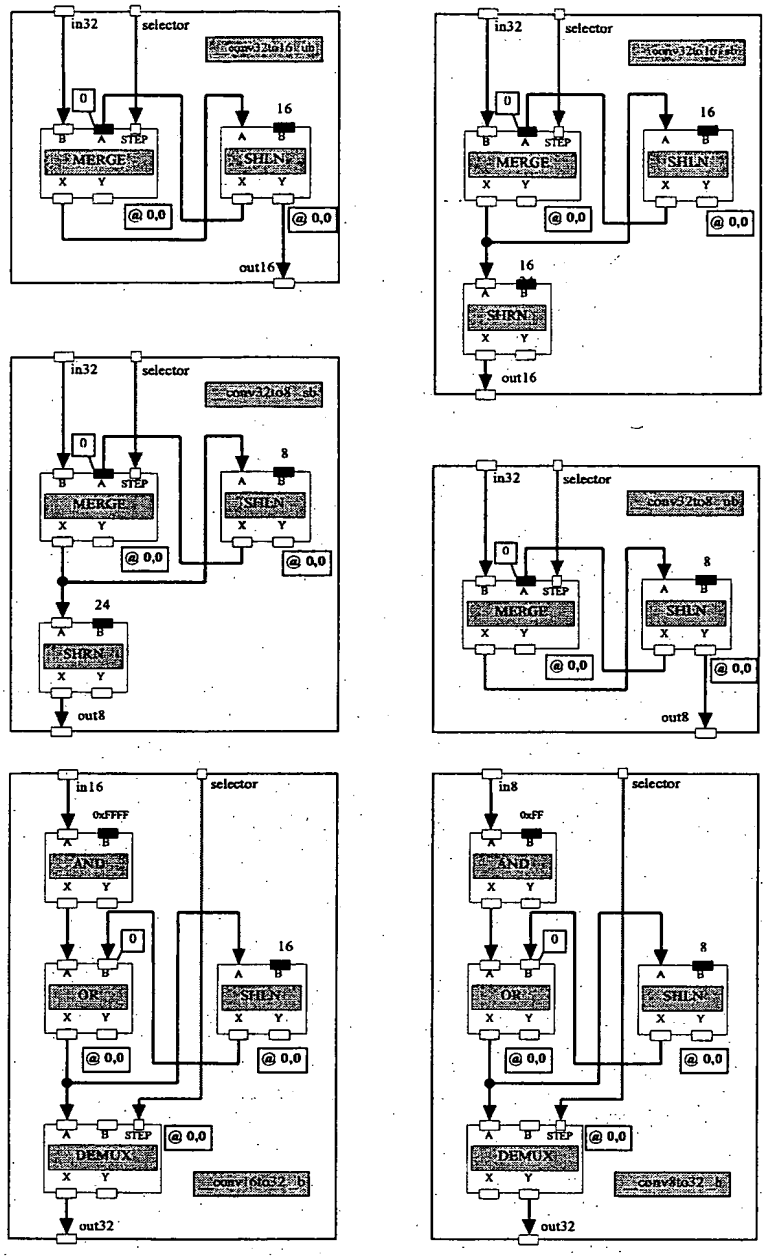


Fig. 14 :

Converter modules for conversion from and to shorter data types. The signed versions suffixed with `_sb` do correct sign extension. All modules 16-bit converters must be connected to `101010..` event streams while the `32to8`-converters must be fed with a `10001000...` sequence and the `8to32` must be fed with an a `00010001...` sequence, respectively. All modules output one packet/cycle.

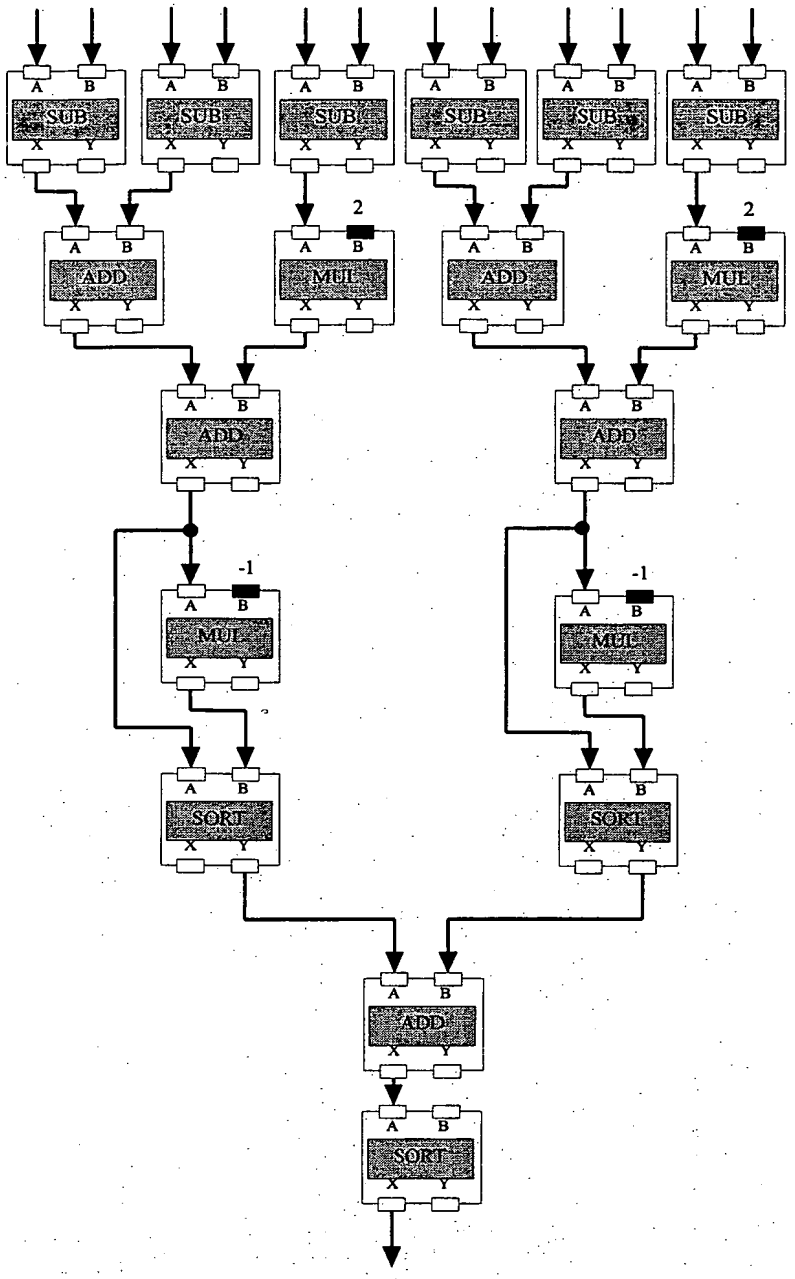


Fig. 15

The main calculation network of the edge3x3 configuration. The MULT-SORT combination does the abs() calculation while the SORT does the min() calculation.

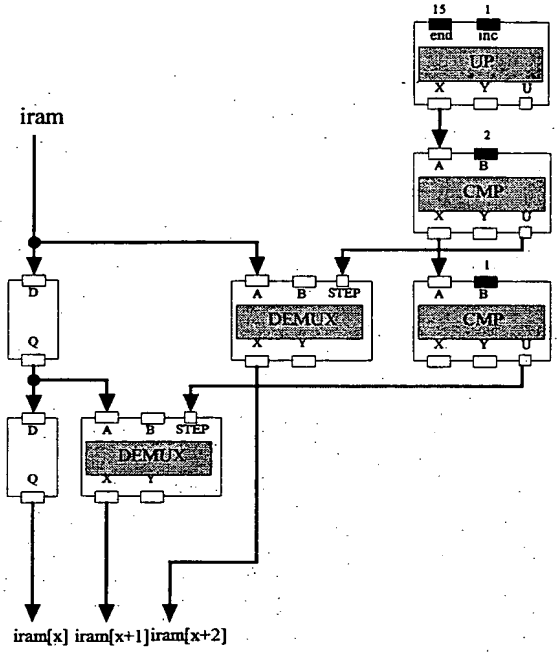


Fig. 16:

Input preparation with shift register synthesis. For each IRAM access one of these modules is generated.

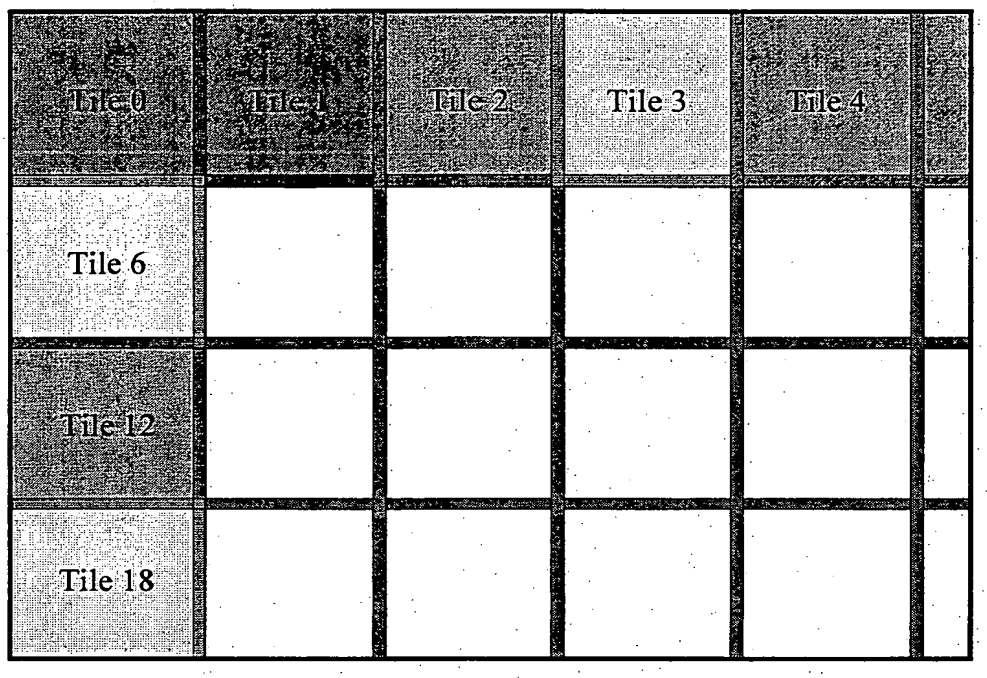


Fig. 17

A sample picture with the size 640 x 480 pixels. Without precautions loop tiling would miss the pixels on the borders between the tiles.

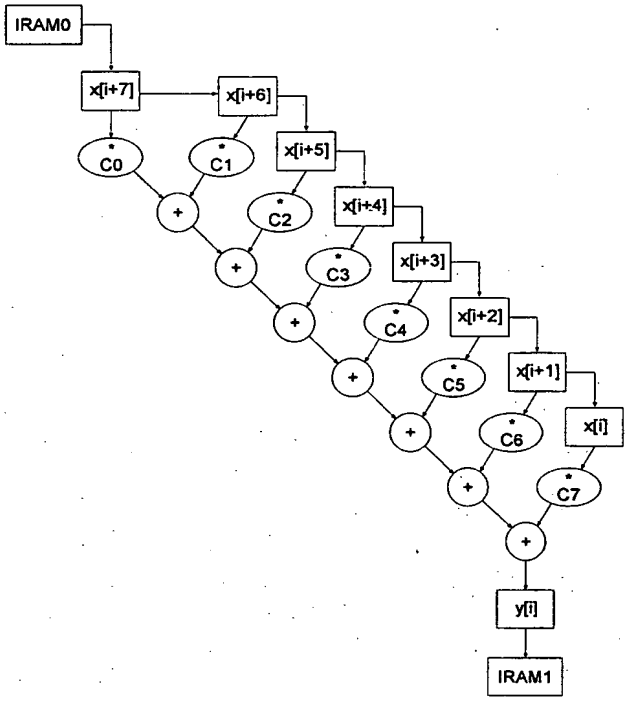

IRAM0
x[i+7]
x[i+6]
x[i+5]
x[i+4]
x[i+3]
x[i+2]
x[i+1]
x[i]
C0
C1
C2
C3
C4
C5
C6
C7
+
y[i]
IRAM1

Fig. 18

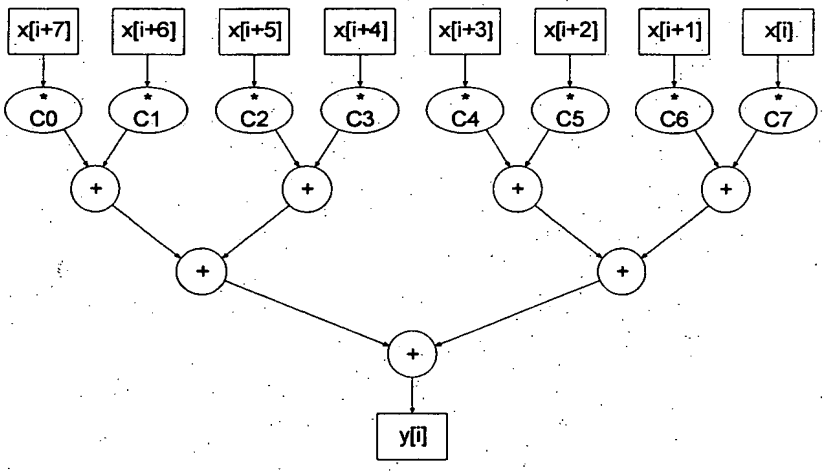

x[i+7]
x[i+6]
x[i+5]
x[i+4]
x[i+3]
x[i+2]
x[i+1]
x[i]
C0
C1
C2
C3
C4
C5
C6
C7
+
y[i]

Fig. 19

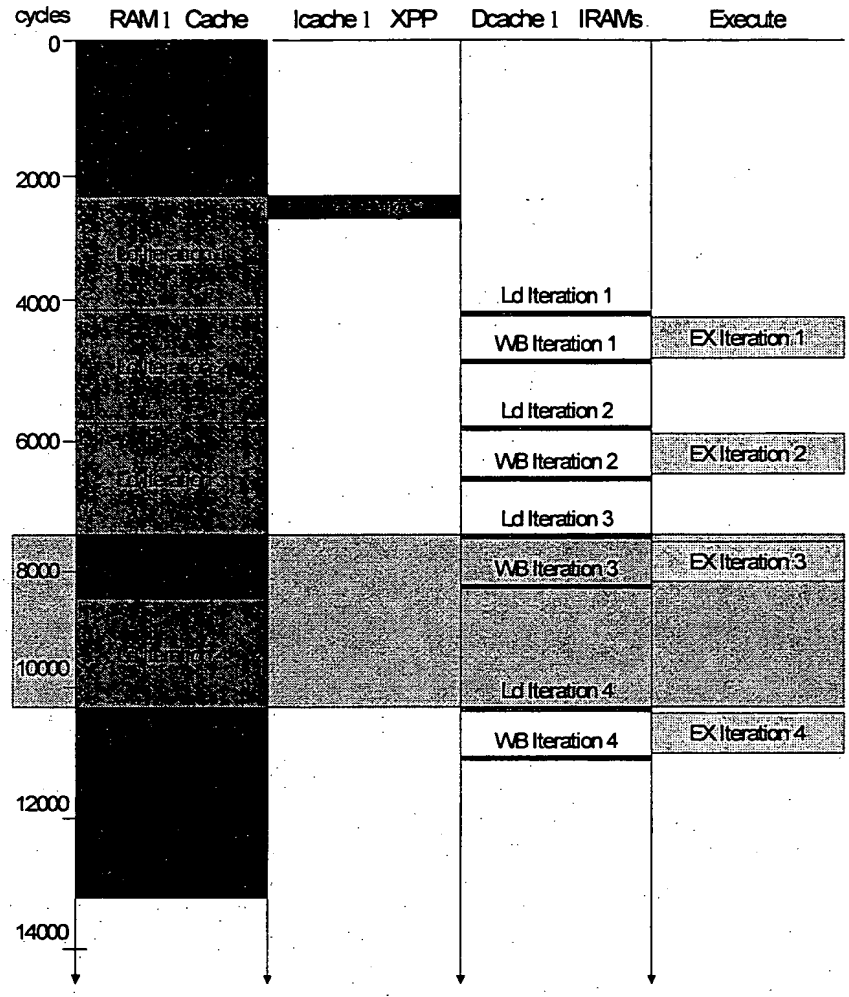

cycles
RAM1 Cache
Icache 1 XPP
Dcache 1 IRAMs
Execute
0
2000
4000
6000
8000
10000
12000
14000
Ld Iteration 1
WB Iteration 1
EX Iteration 1
Ld Iteration 2
WB Iteration 2
EX Iteration 2
Ld Iteration 3
WB Iteration 3
EX Iteration 3
Ld Iteration 4
WB Iteration 4
EX Iteration 4

Fig. 20

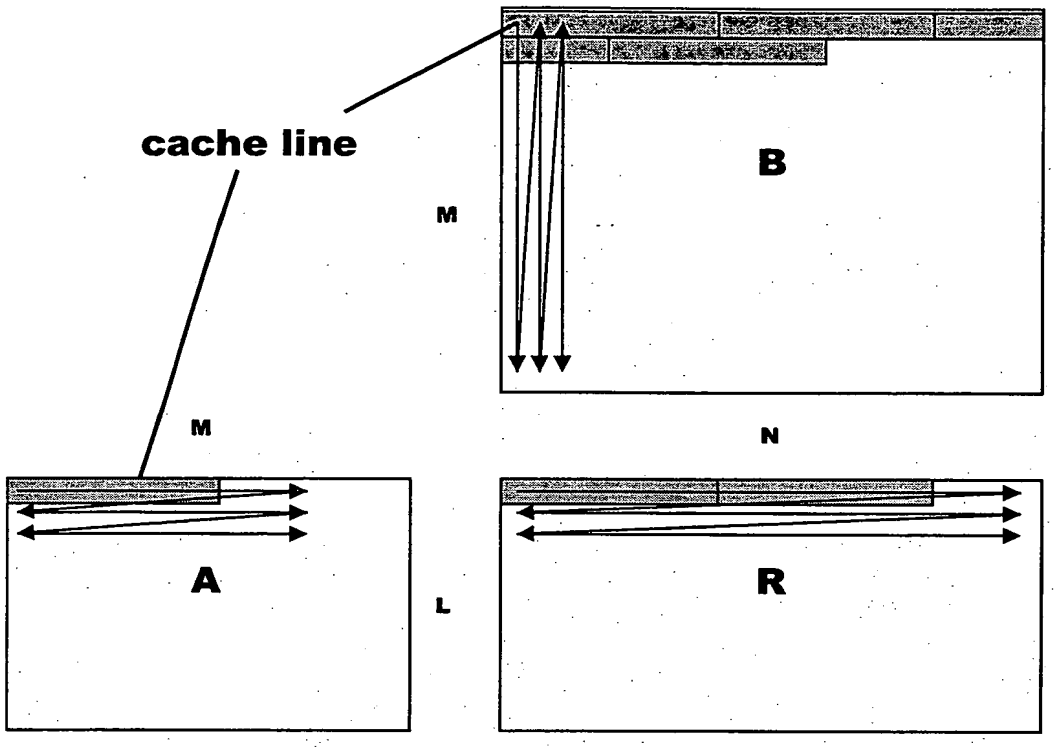


Fig. 21: The visualized array access sequences.

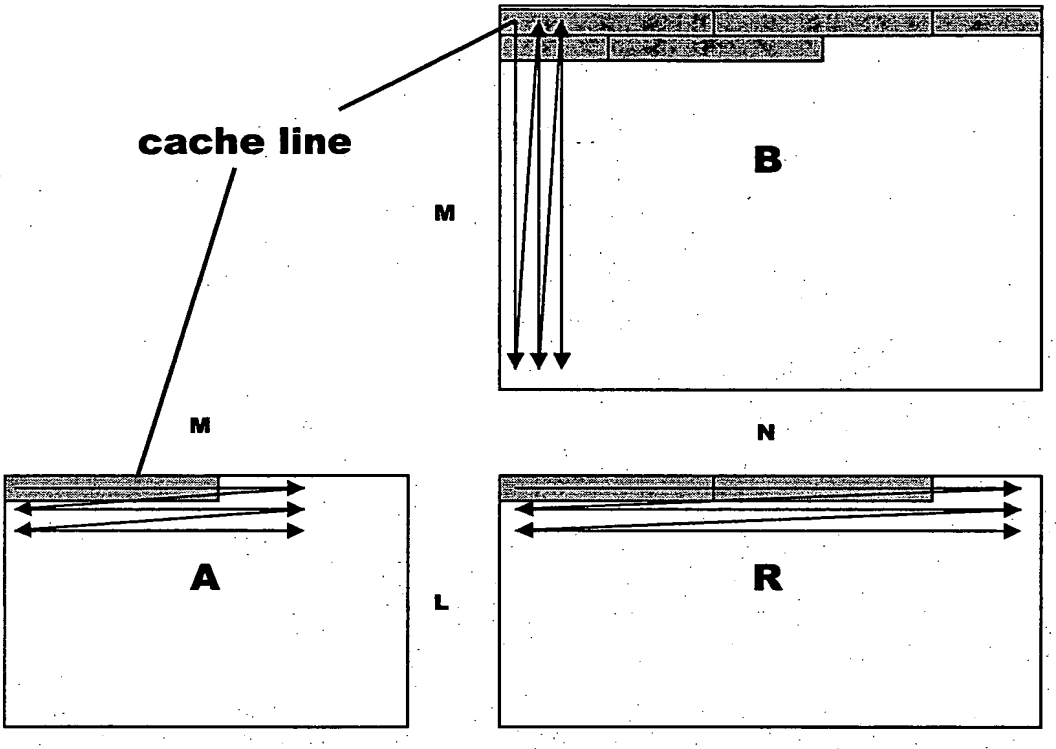


Fig. 22:

The visualized array access sequences after optimization. Here the improvement is evident, since array B is now read following the cache lines.

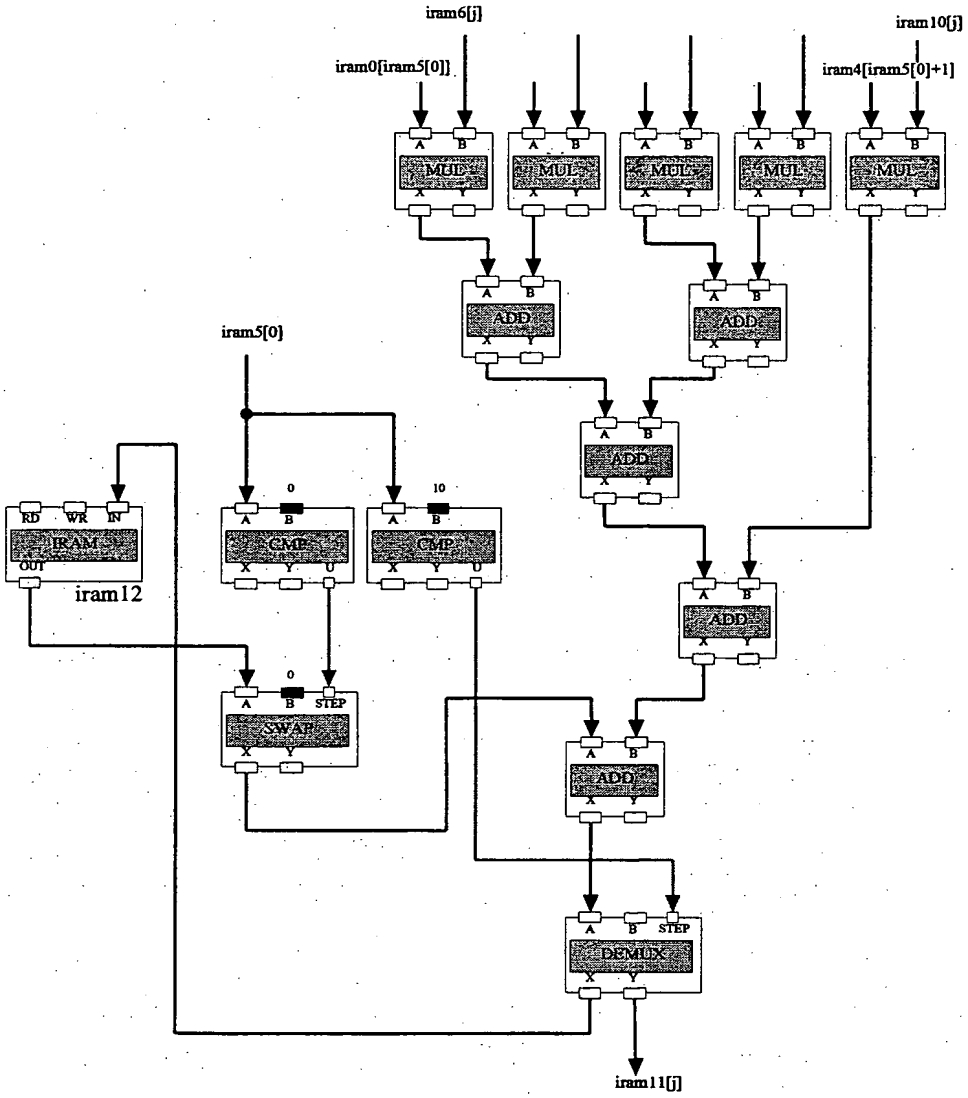


Fig. 23:

Dataflow graph of matrix multiplication after unroll-and-jam. Counters and address calculations are omitted.

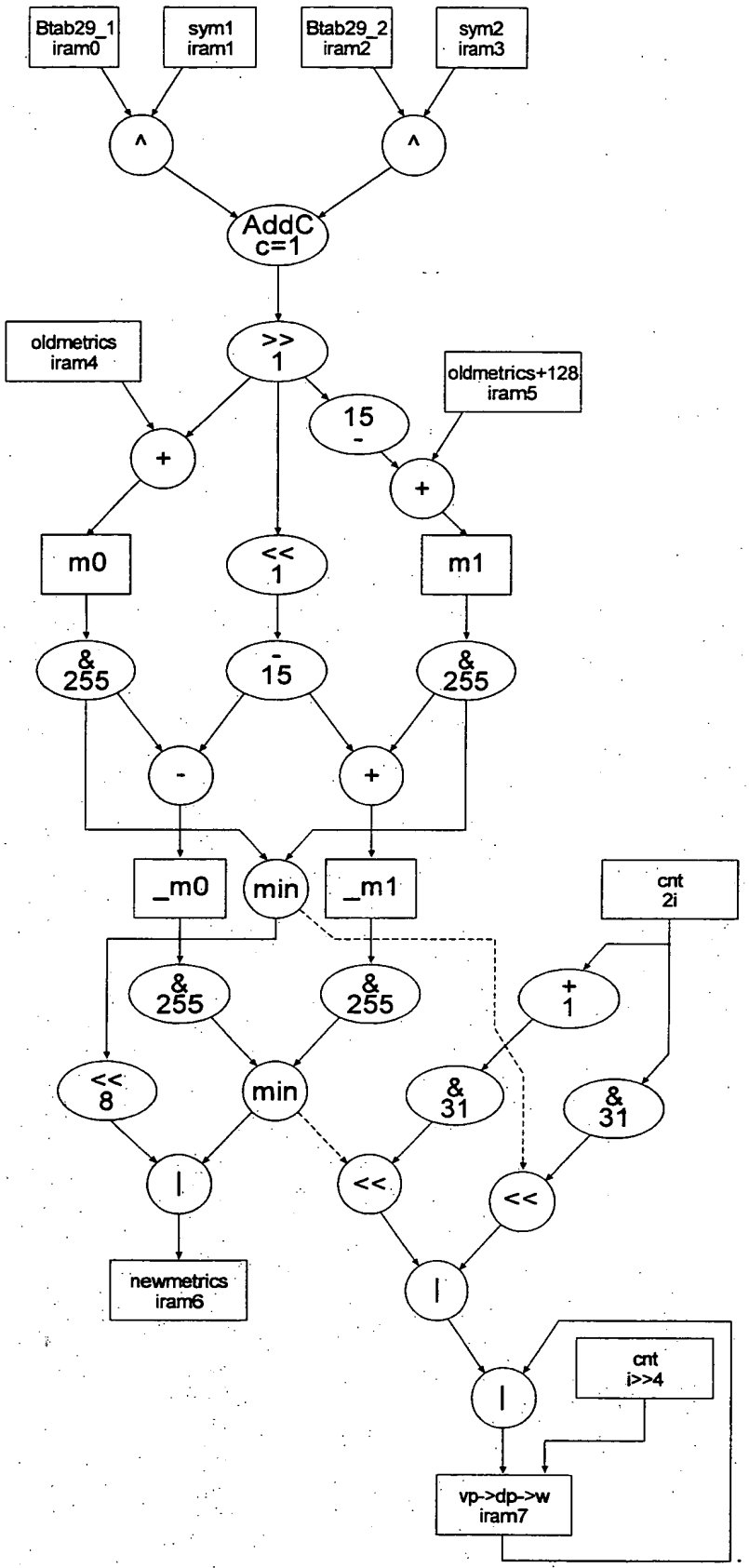

Btab29_1
iram0
sym1
iram1
Btab29_2
iram2
sym2
iram3
^
^
AddC
c=1
oldmetrics
iram4
>>
1
oldmetrics+128
iram5
15
-
+
+
m0
<<
1
m1
&
255
-
15
&
255
-
+
_m0
min
_m1
cnt
2i
&
255
&
255
+
1
<<
8
min
&
31
&
31
|
<<
<<
newmetrics
iram6
|
cnt
i>>4
|
vp->dp->w
iram7

Fig. 24

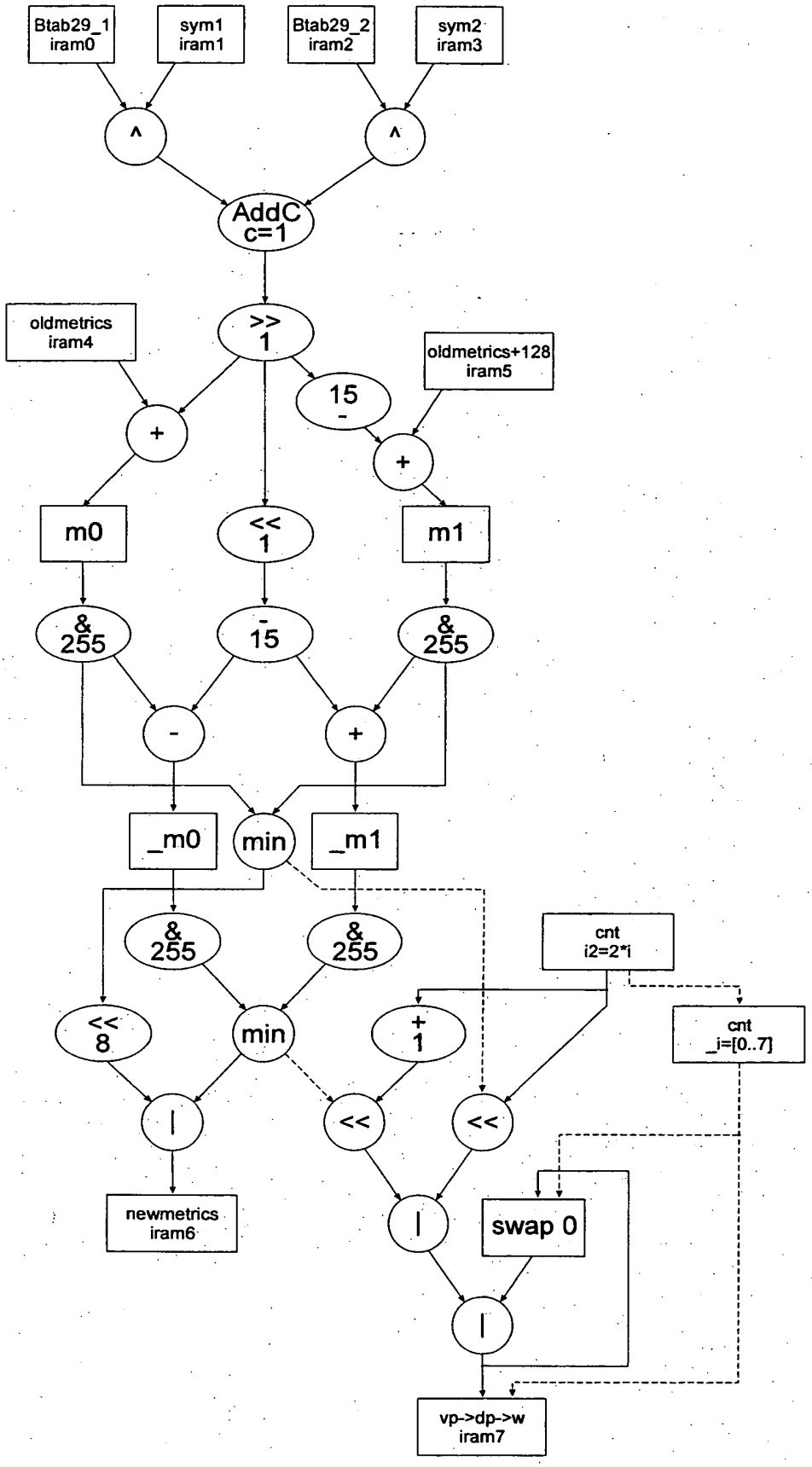


Fig. 25: The modified dataflow graph, where unrolling and splitting have been omitted for simplicity

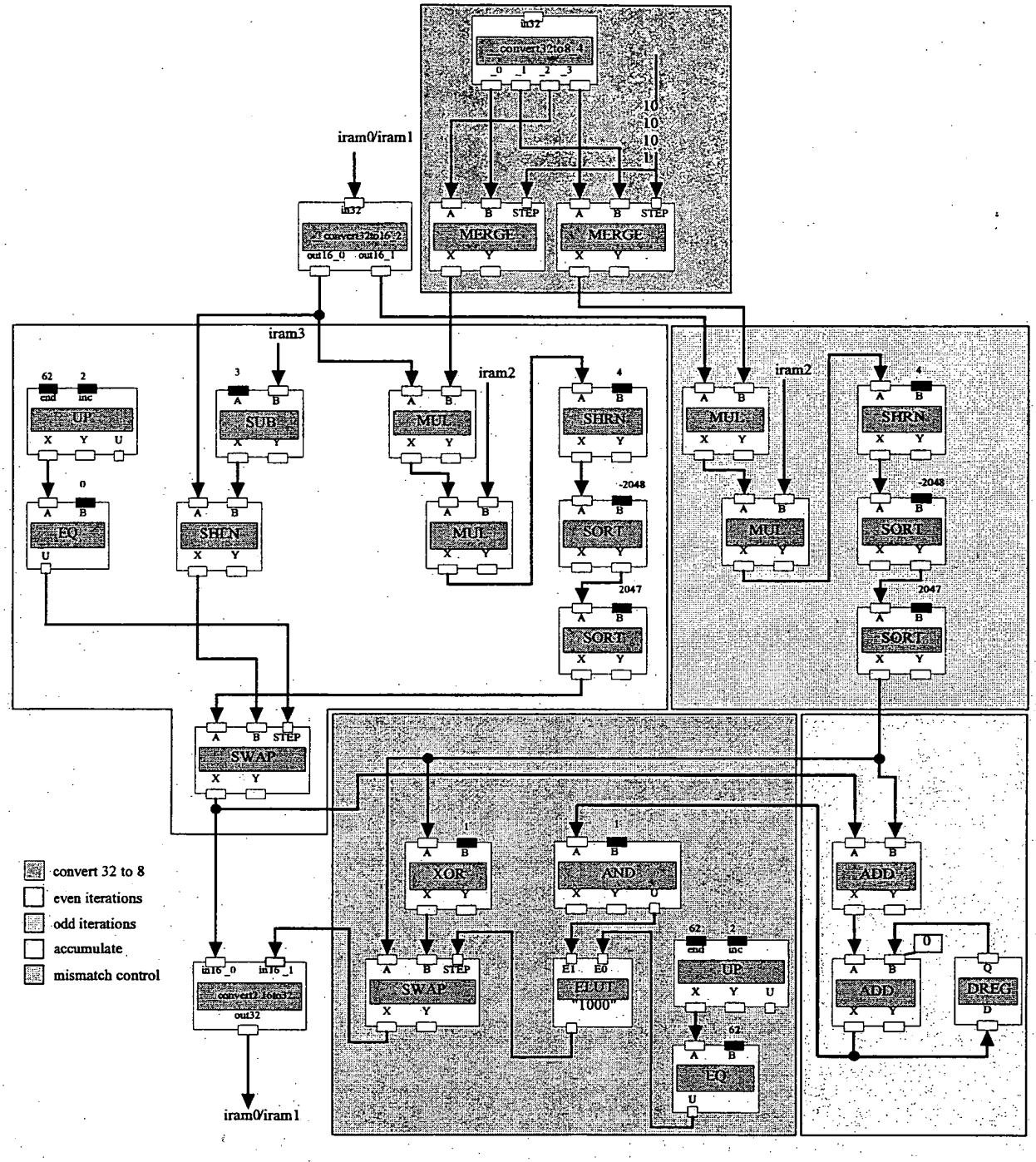


Fig. 26:

Dataflow graph of the MPEG2 inverse quantization for intra coded blocks. The yellow and green blocks were produced by partial unrolling. The difference is that the green block must no account for the special iteration value 0. The blue block does the accumulation which alters the value at iteration 64 if necessary.

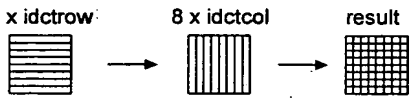


Fig. 27

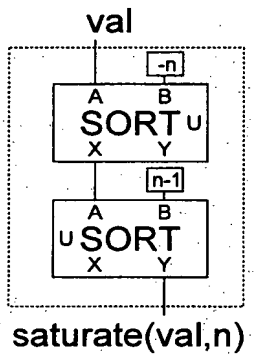


Fig. 28

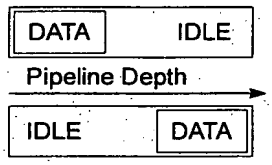


Fig. 29

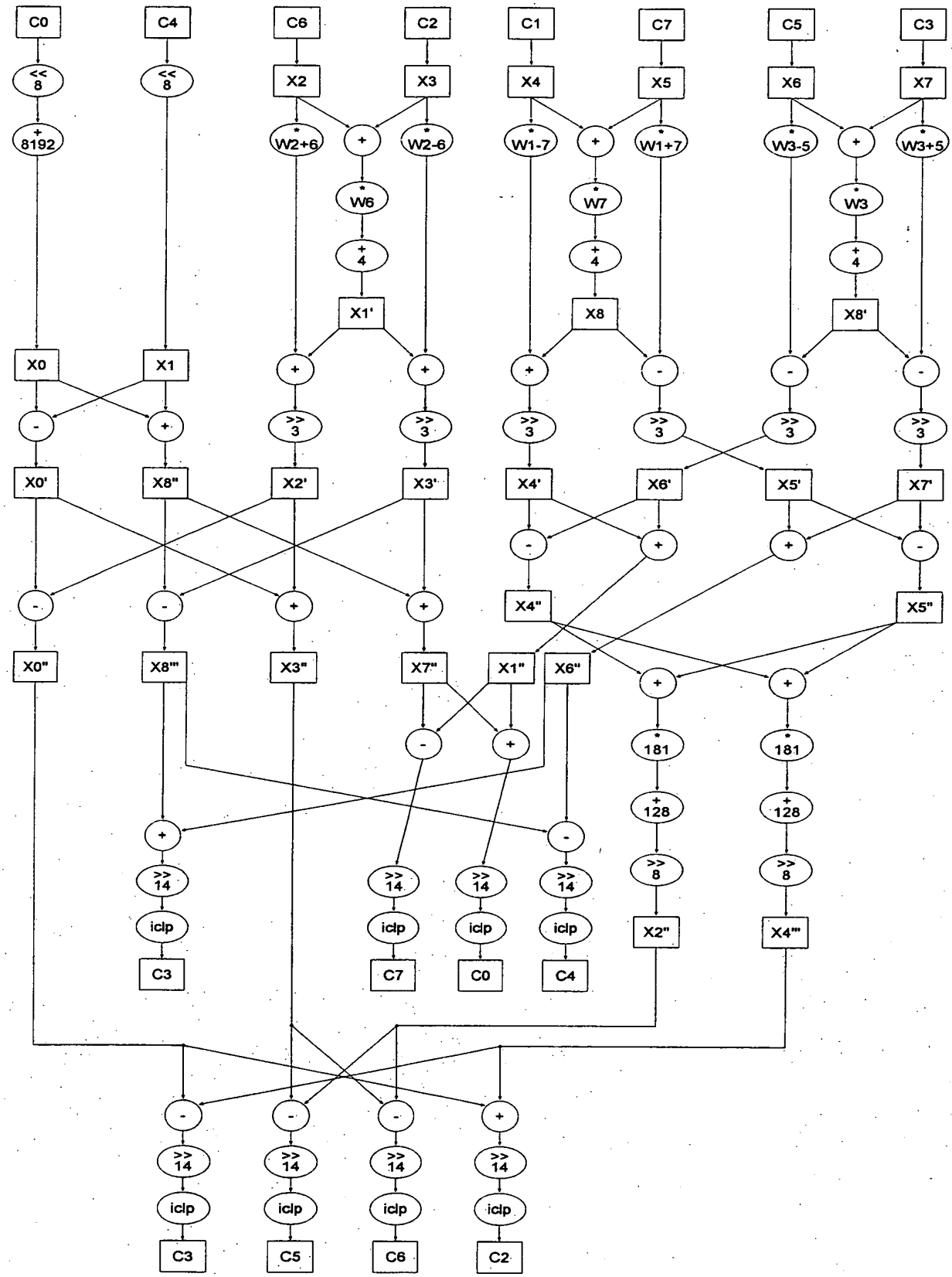


Fig. 30: Dataflow Graph of idct column processing

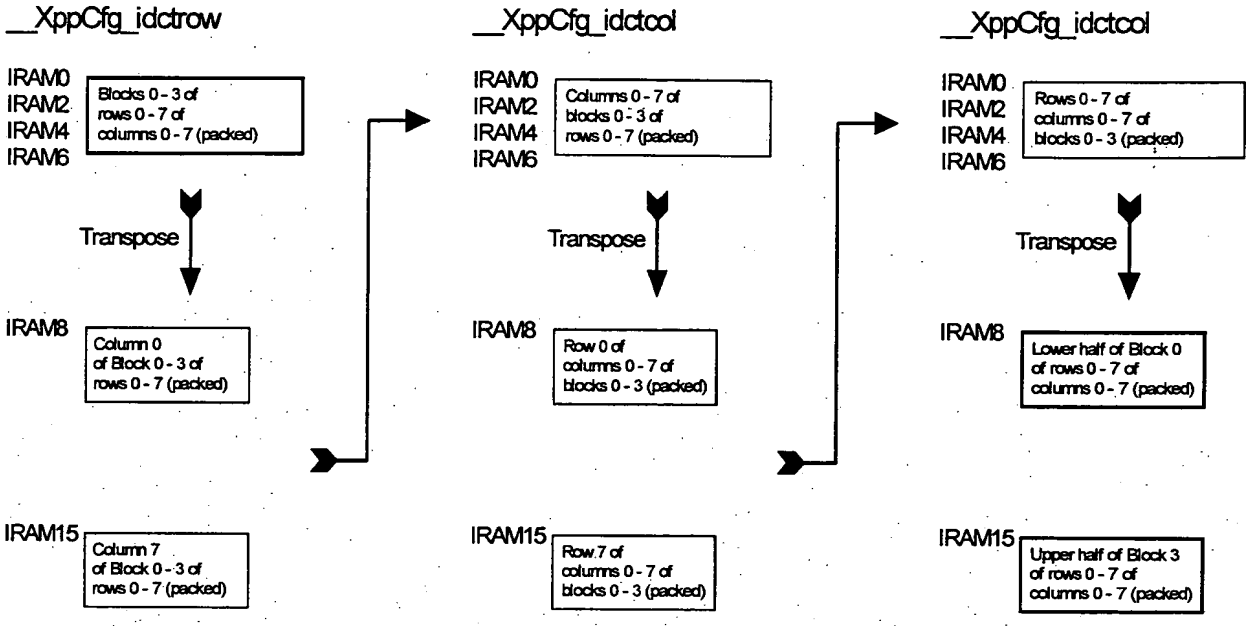


Fig. 31: Data layout transformations in idct configurations

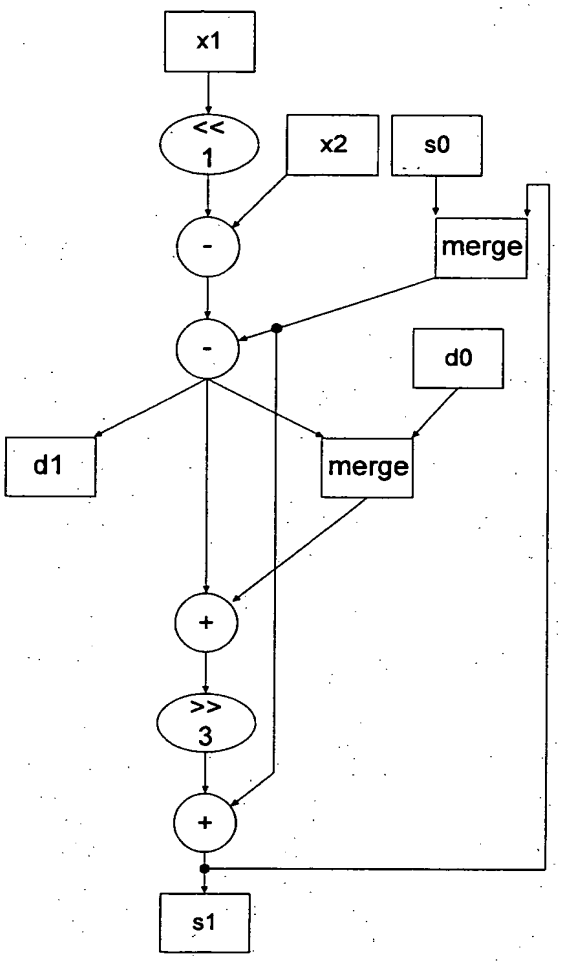


Fig. 32: Dataflow graph of the innermost loop nest.

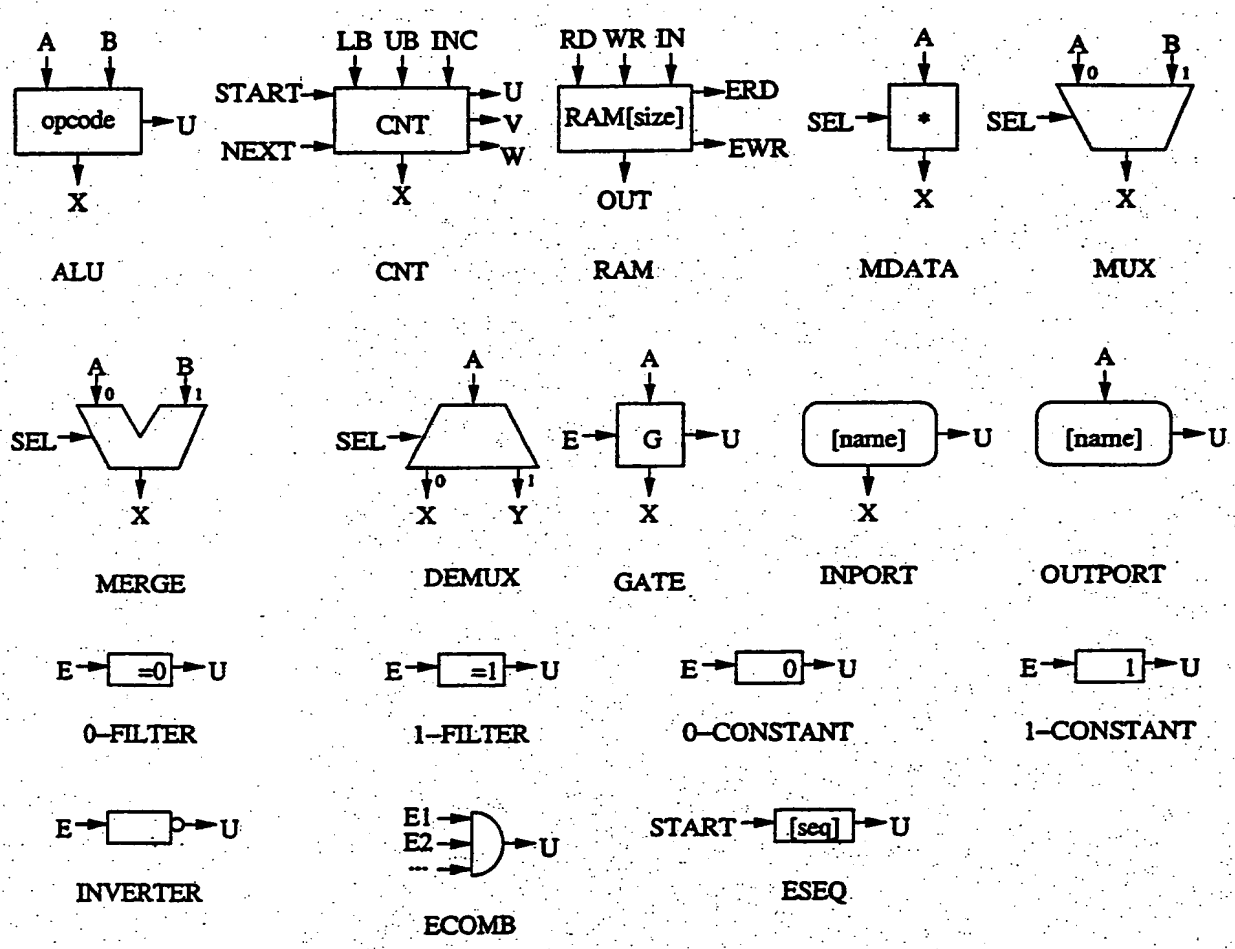


Fig. 33: Functions of an RDFP

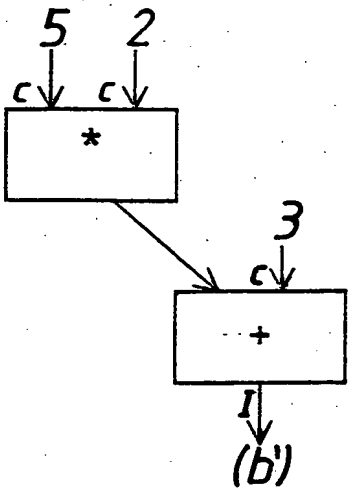


Fig. 34

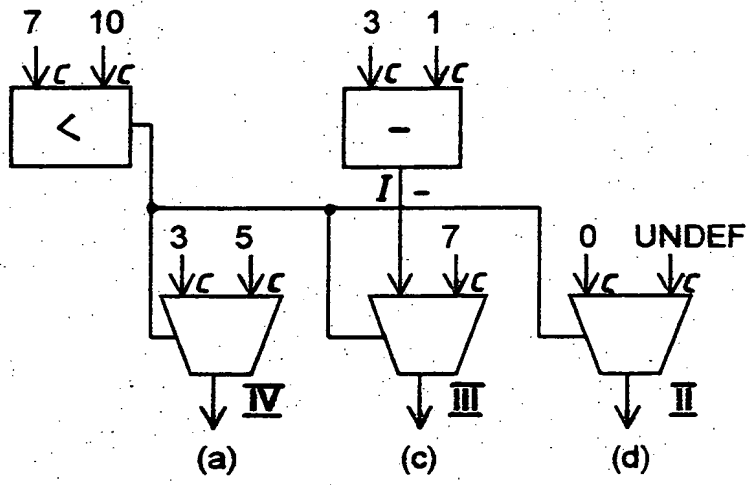


Fig. 35

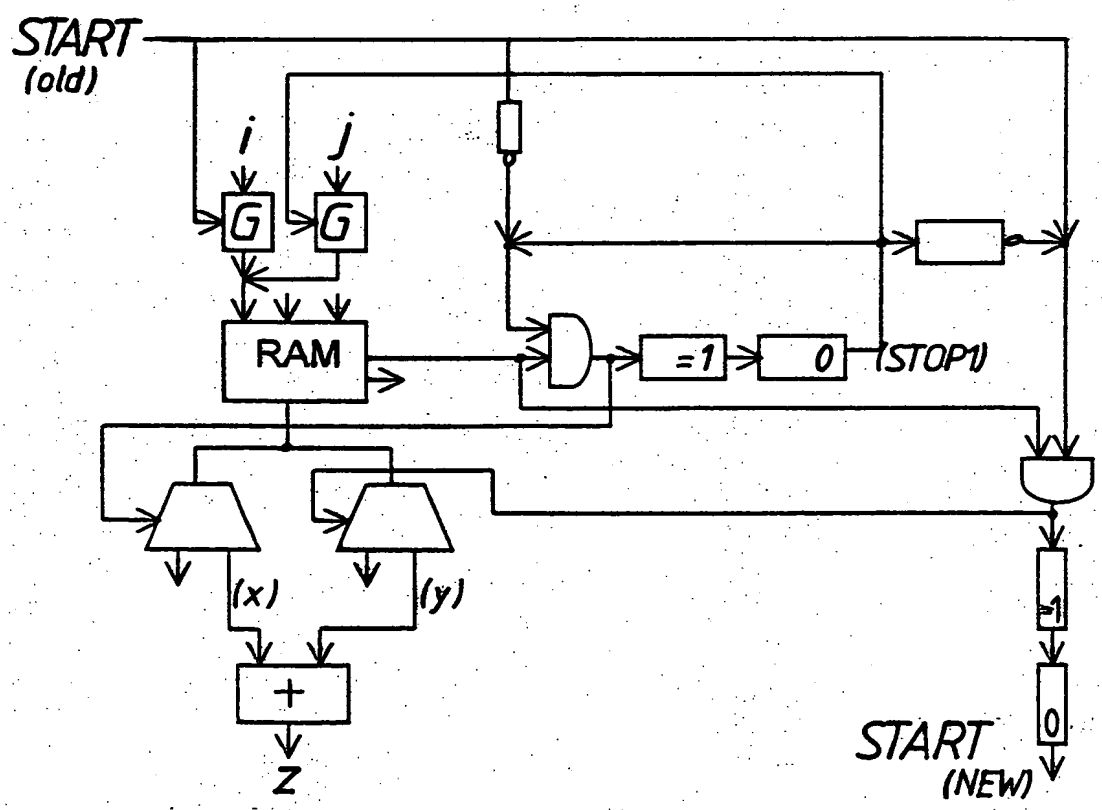


Fig. 36

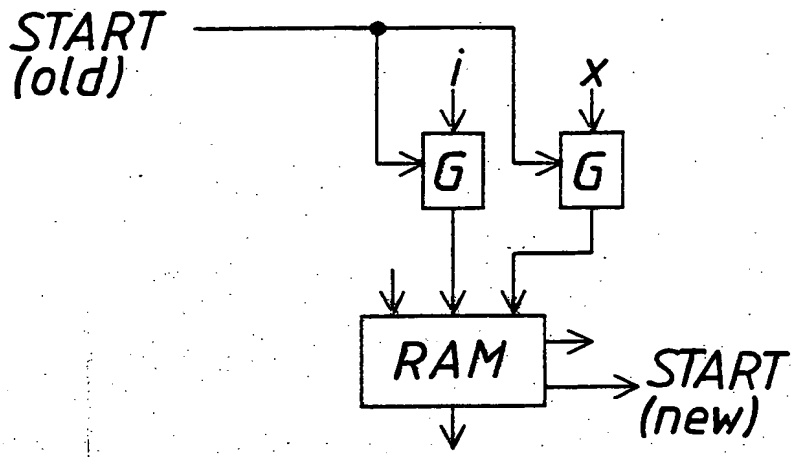

START
(old)
i
x
G
G
RAM
START
(new)

Fig. 37

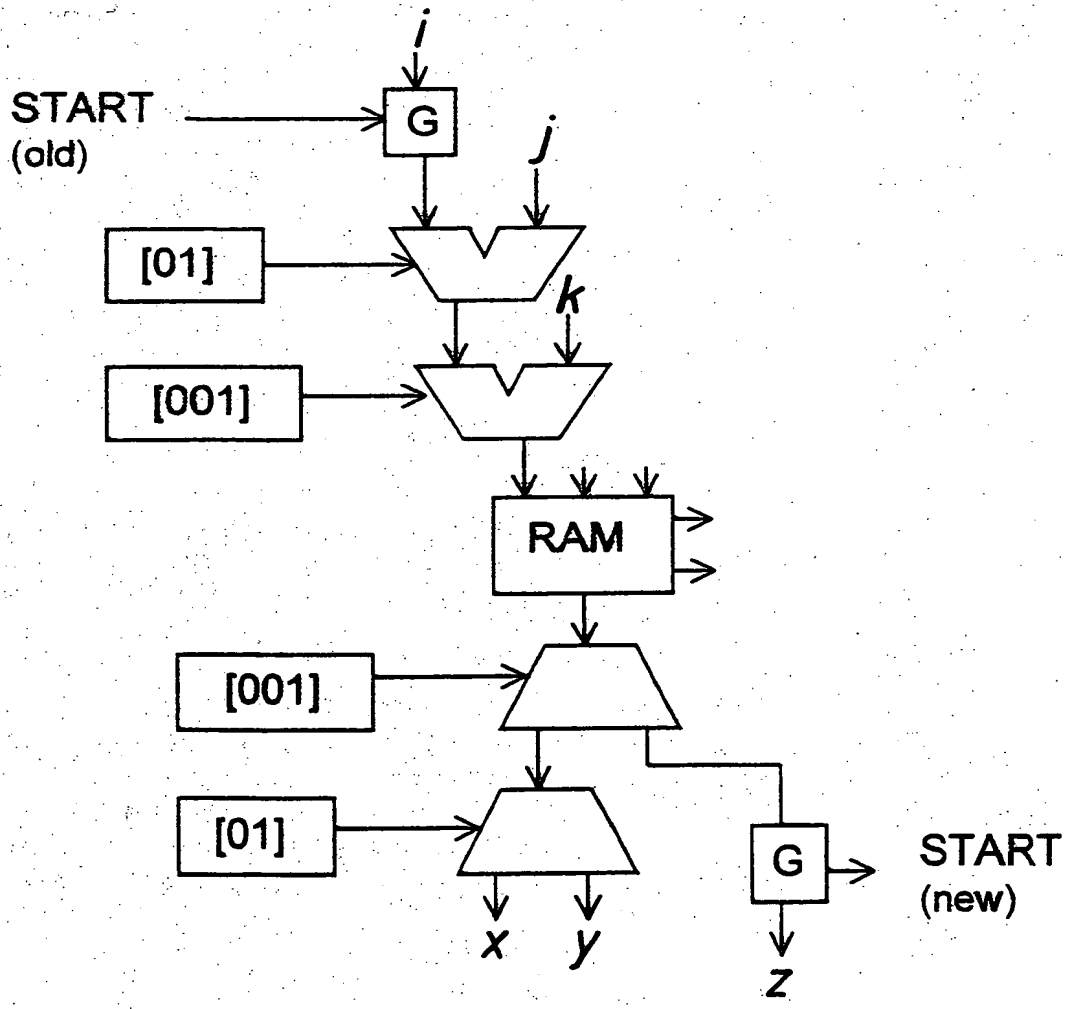

i
START
(old)
G
j
[01]
k
[001]
RAM
[001]
[01]
G
START
(new)
x
y
z

Fig. 38

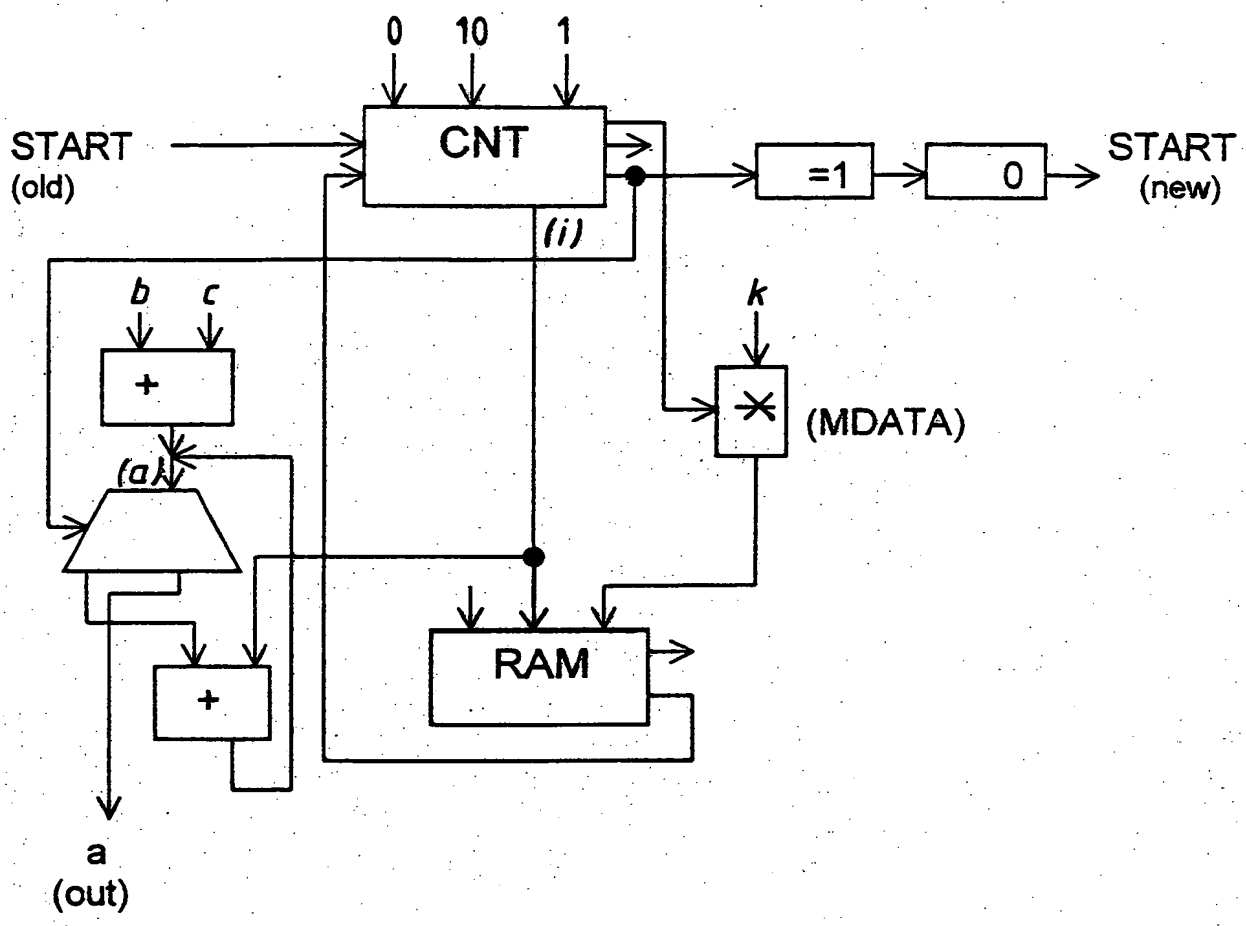


Fig. 39

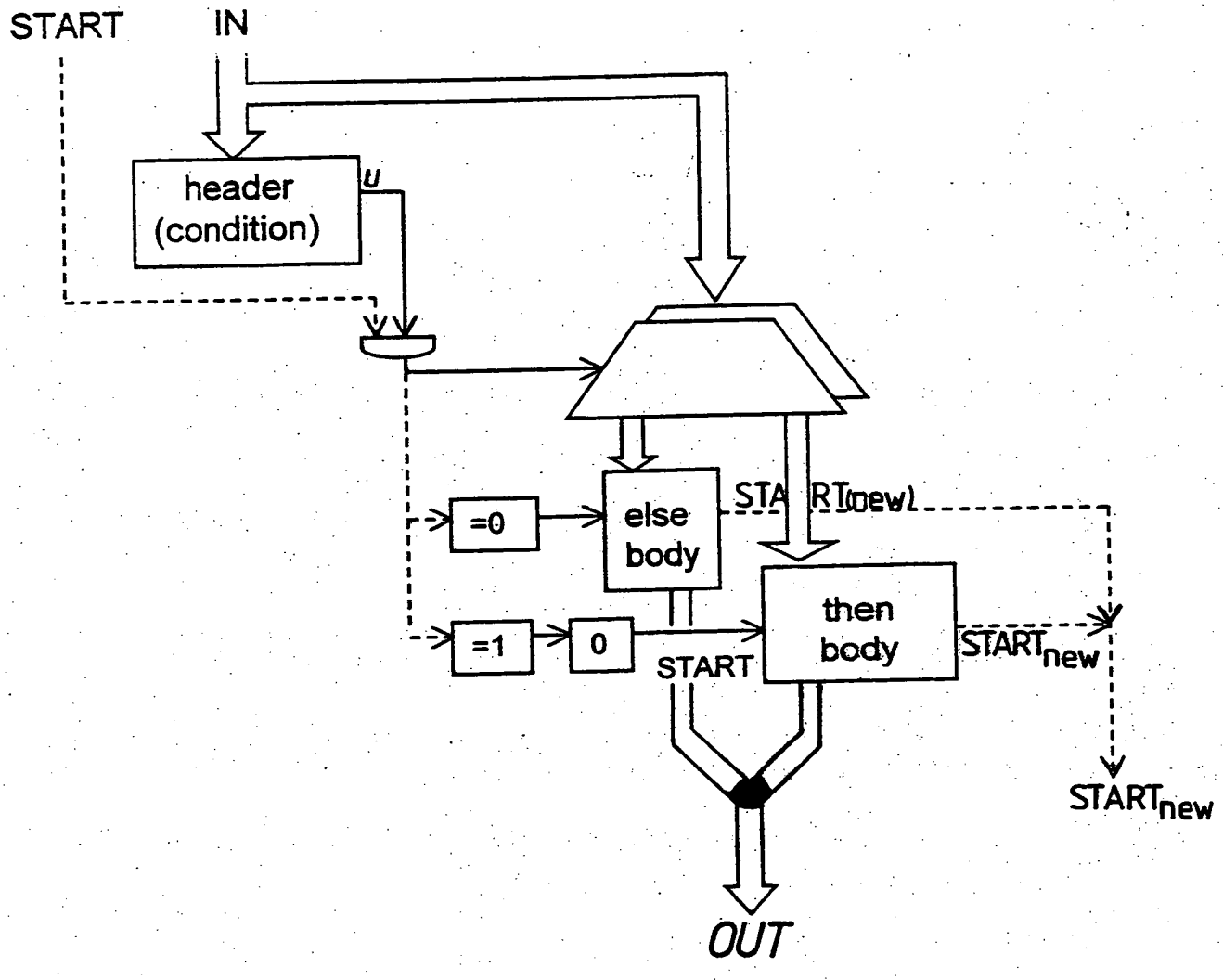


Fig.40: General Conditional Statement Template

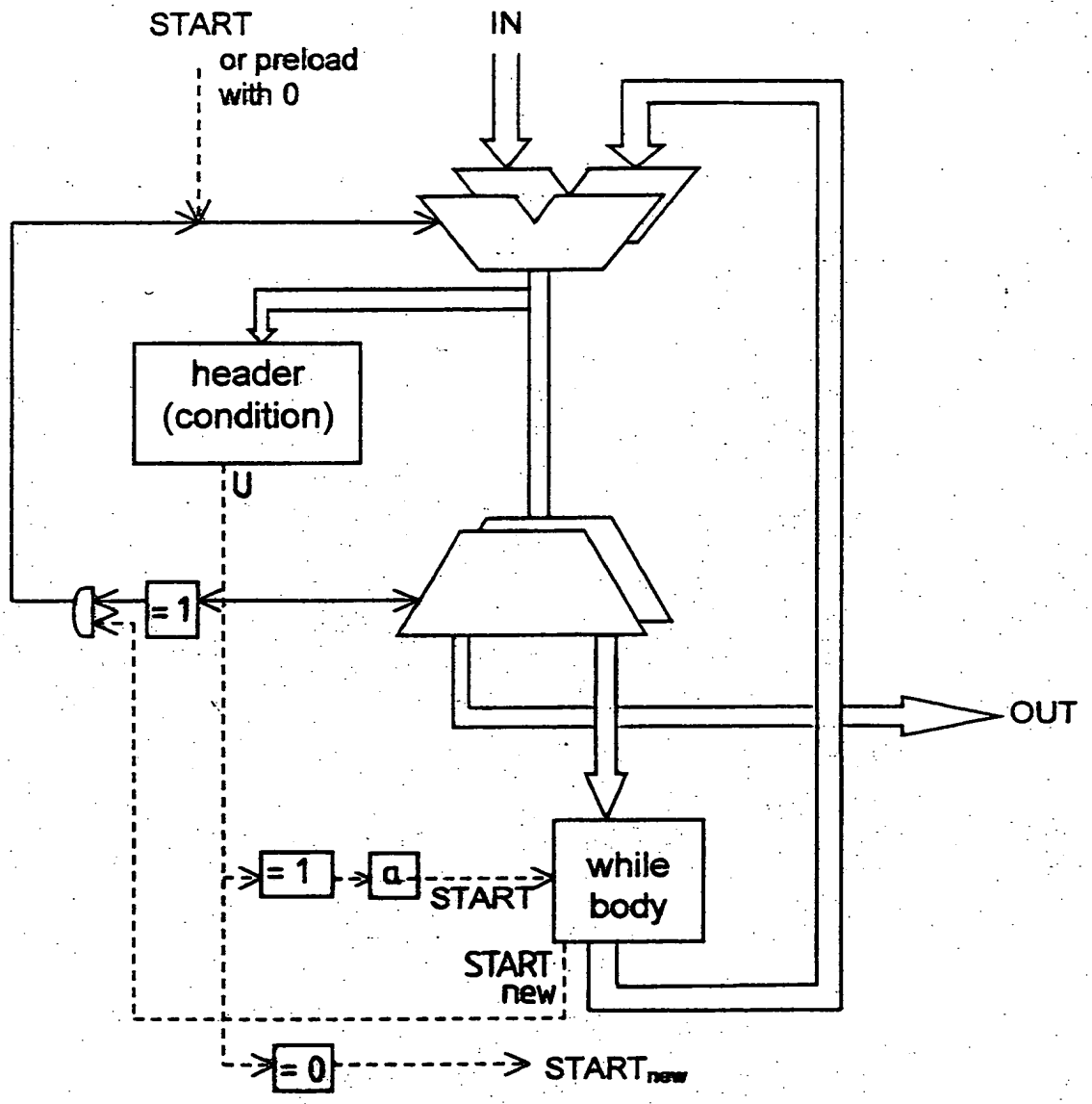


Fig. 44: While Loop Template

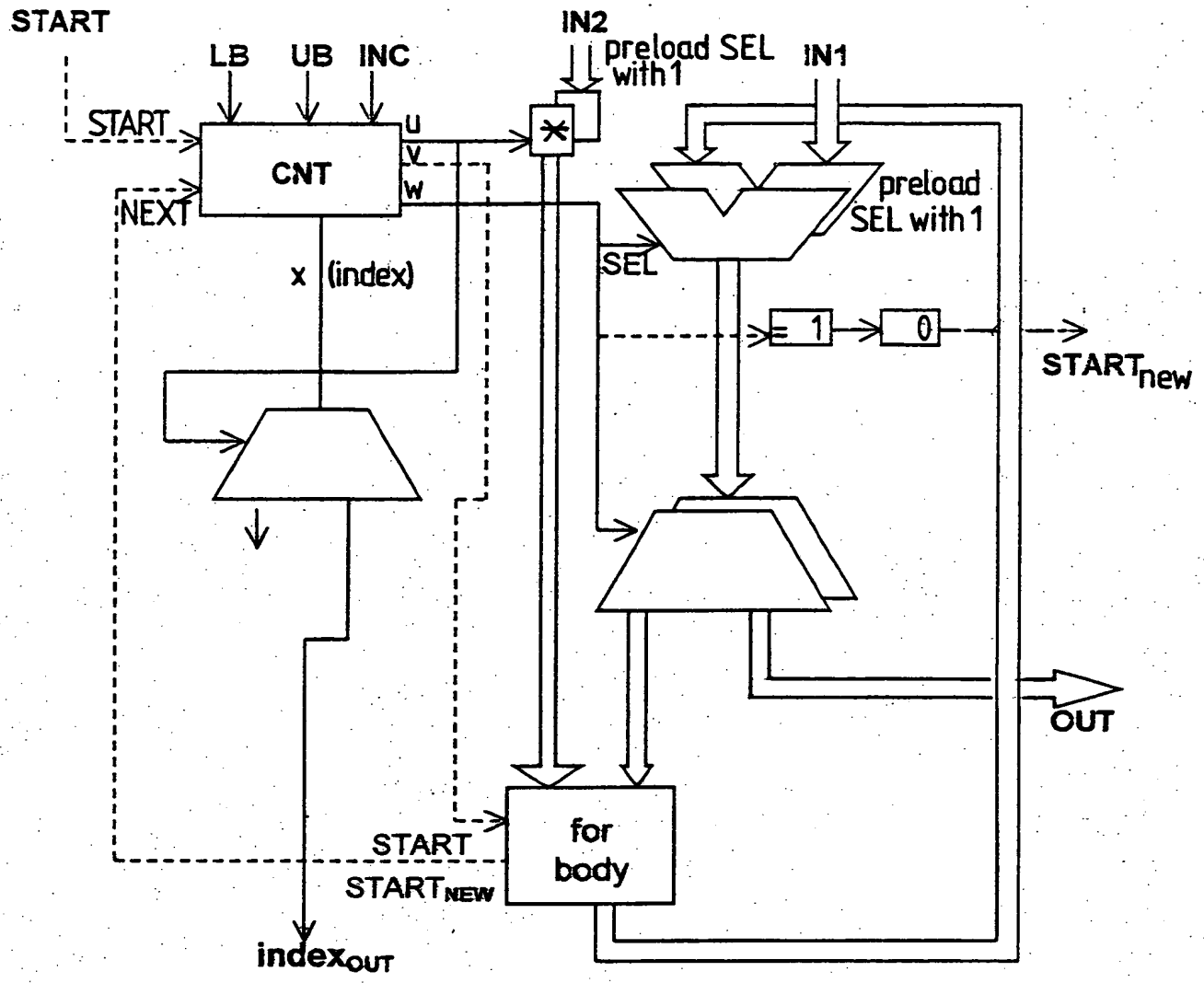


Fig. 42: For Loop Template

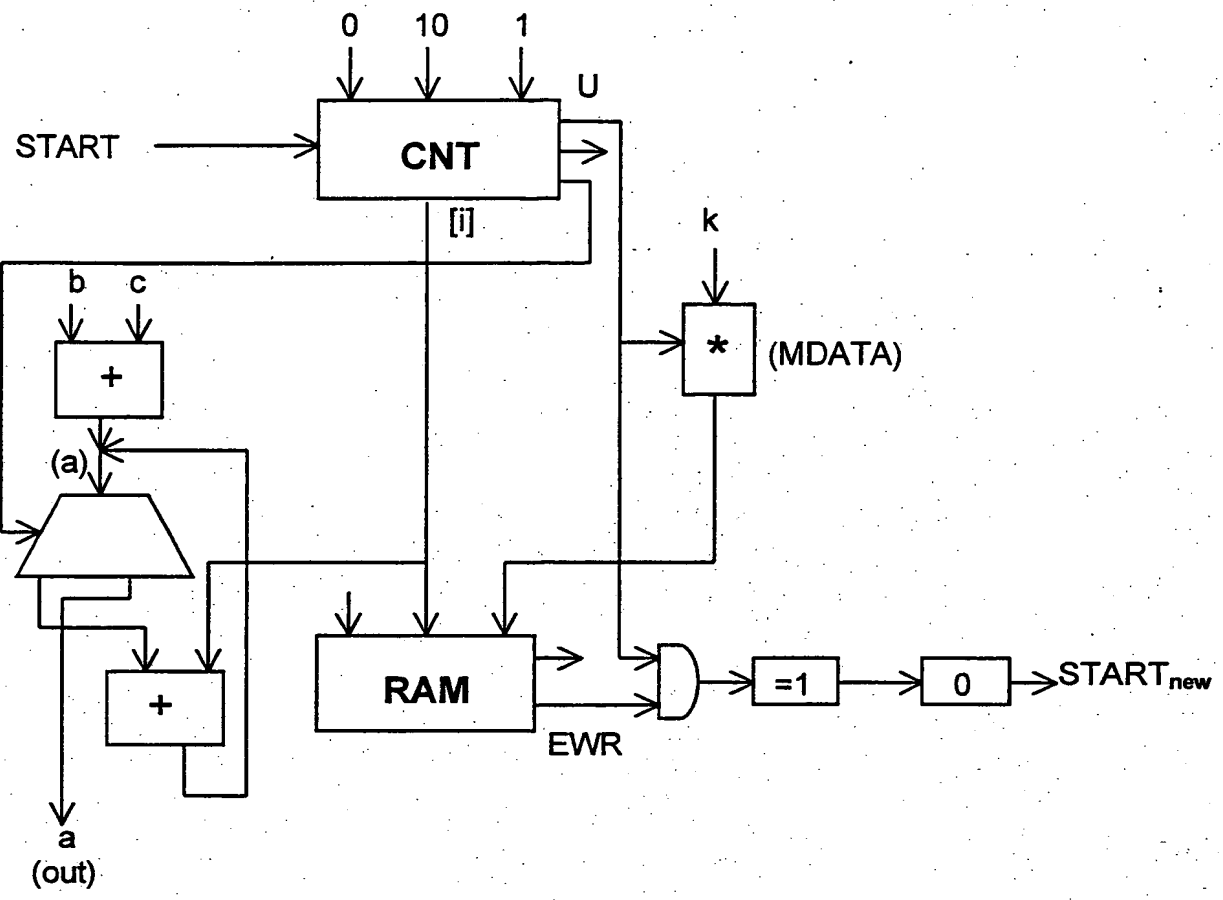


Fig. 43

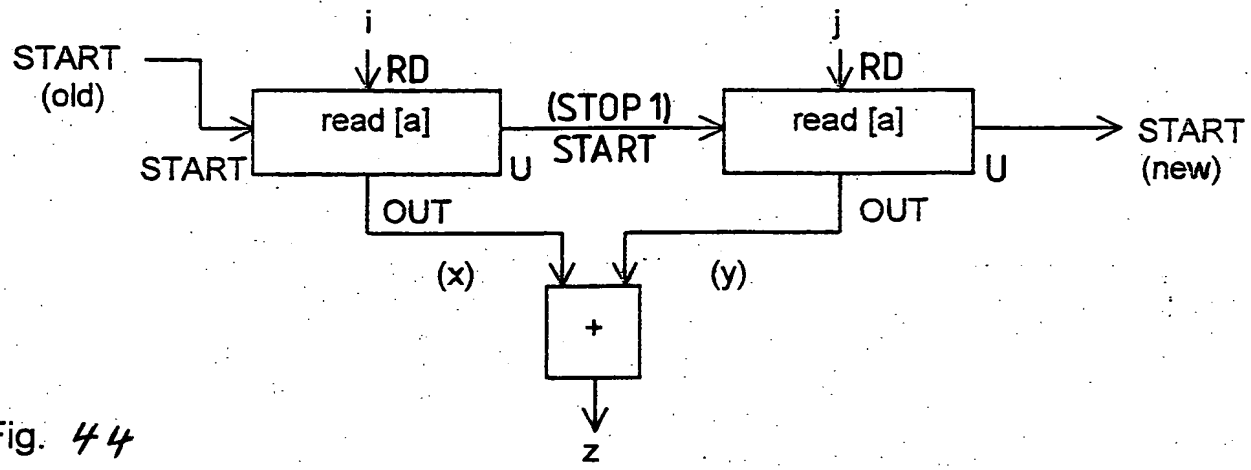

START
(old)
START
i
RD
read [a]
U
(STOP 1)
START
j
RD
read [a]
U
START
(new)
OUT
OUT
(x)
(y)
+
z

Fig. 44

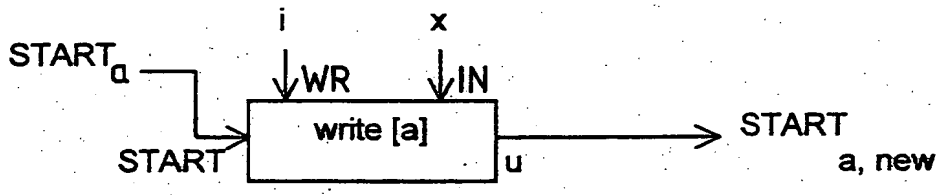

START
a
START
i
WR
x
IN
write [a]
u
START
a, new

Fig. 45

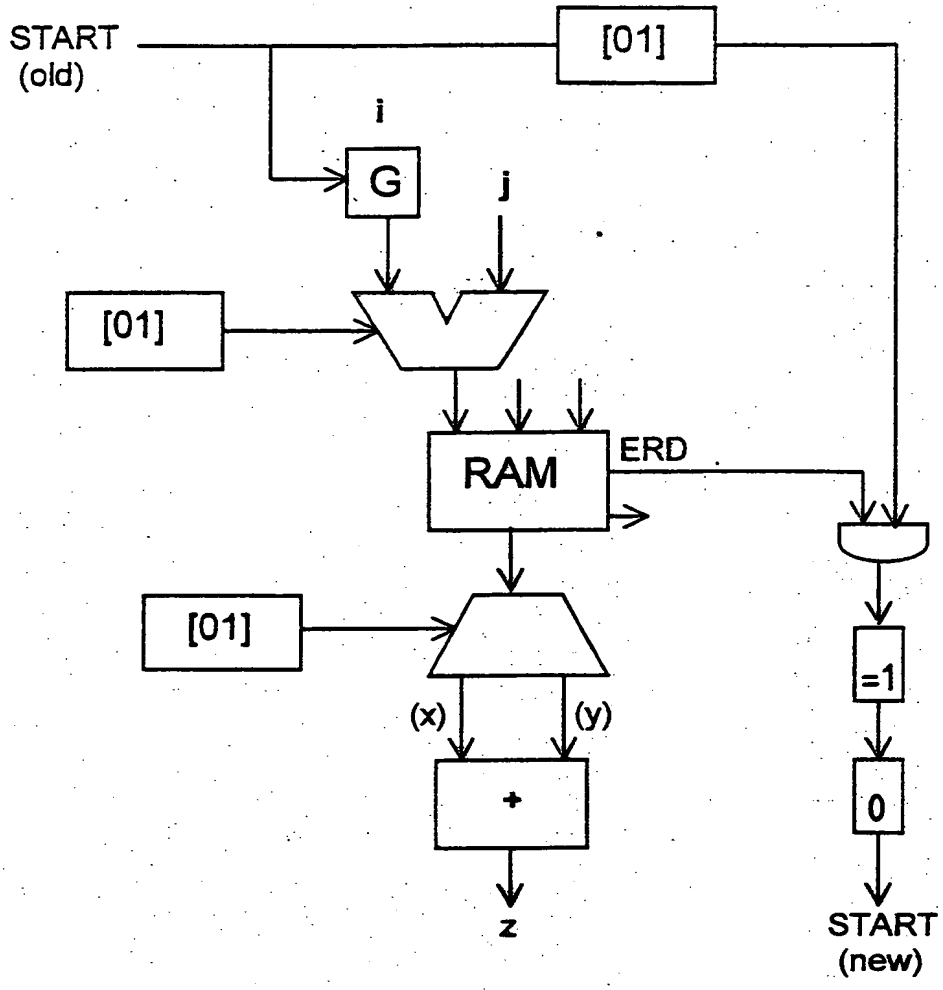


Fig. 46

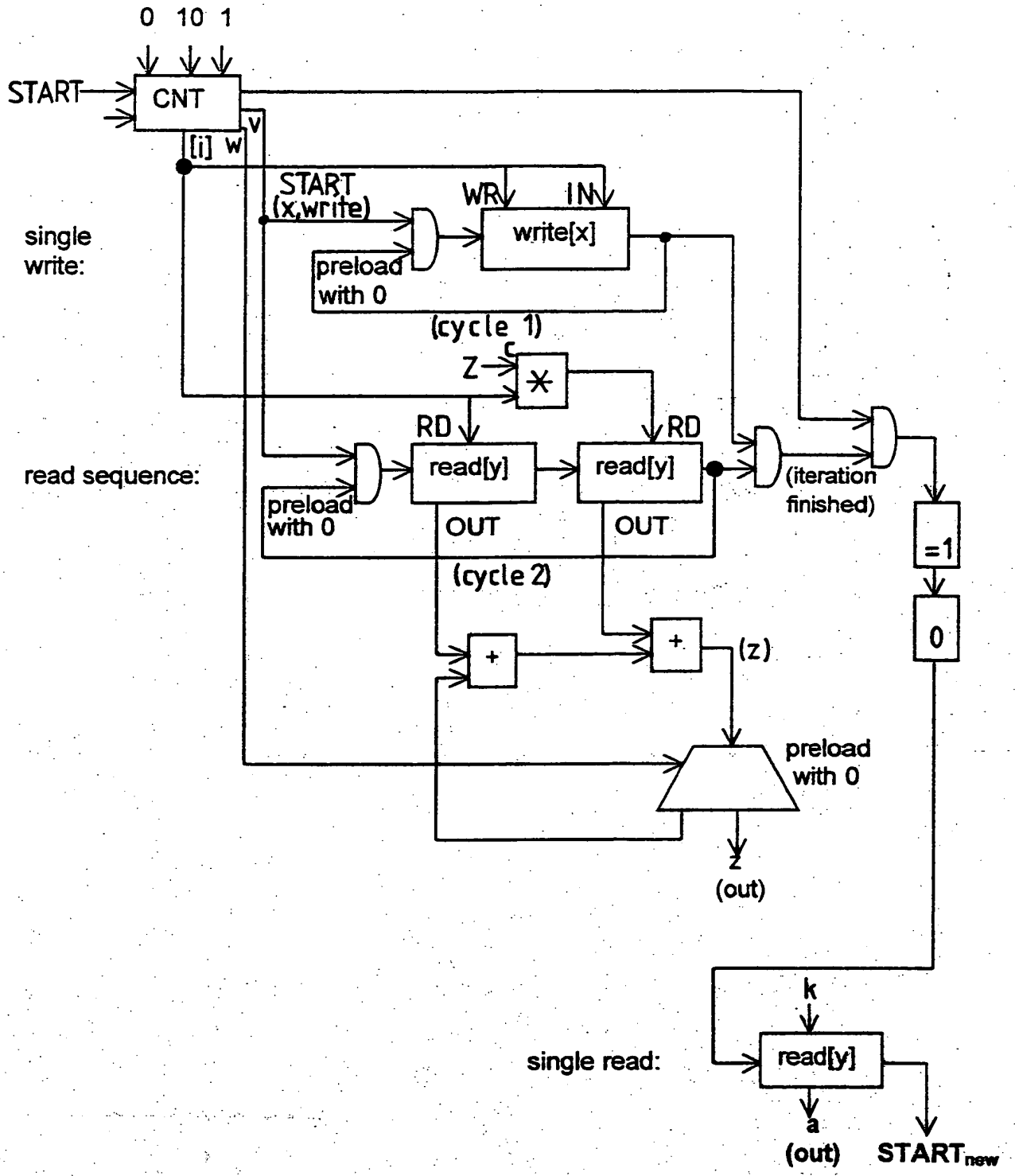

0 10 1
START
CNT
v
[i] w
START
(x,write)
WR
IN
write[x]
single
write:
preload
with 0
(cycle 1)
Z
RD
RD
read sequence:
read[y]
read[y]
preload
with 0
OUT
OUT
(iteration
finished)
=1
0
(cycle 2)
+
+
(z)
preload
with 0
z
(out)
k
single read:
read[y]
a
(out)
START new

Fig. 47

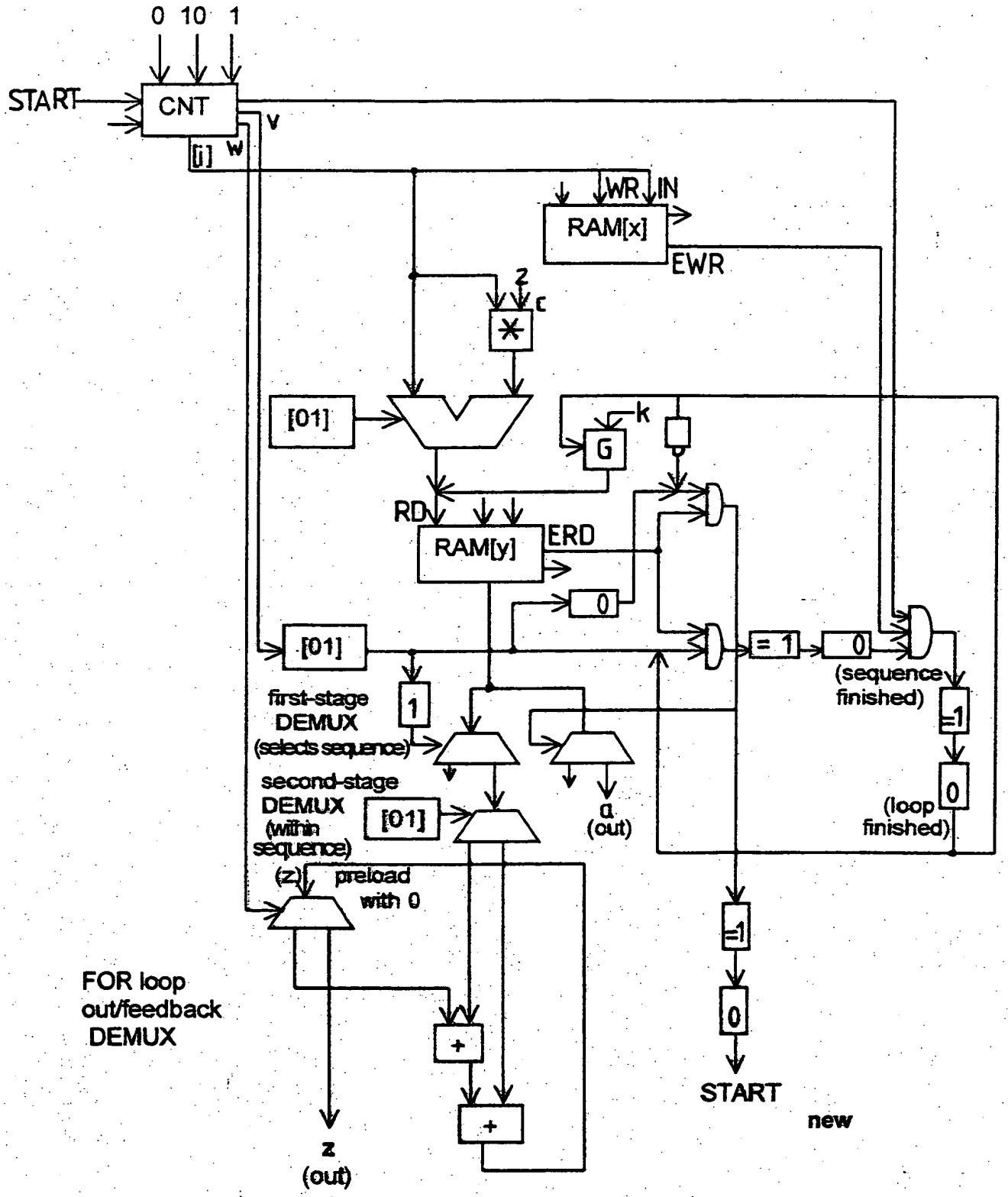

0 10 1
START
CNT
v
[j] w
WR IN
RAM[x]
EWR
z
c
*
[01]
k
G
RD
RAM[y]
ERD
0
[01]
= 1
0
(sequence finished)
=1
0
(loop finished)
first-stage DEMUX (selects sequence)
1
second-stage DEMUX (within sequence)
[01]
a (out)
(z)
preload with 0
FOR loop out/feedback DEMUX
+
+
z (out)
=1
0
START new

Fig. 48

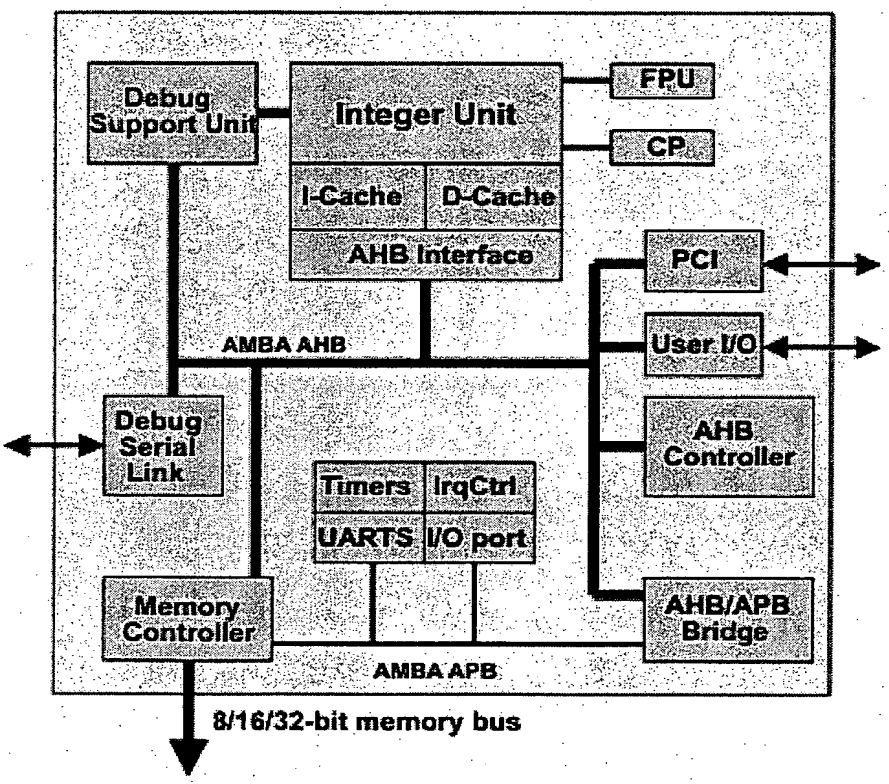


Fig. 49: LEON Architecture Overview

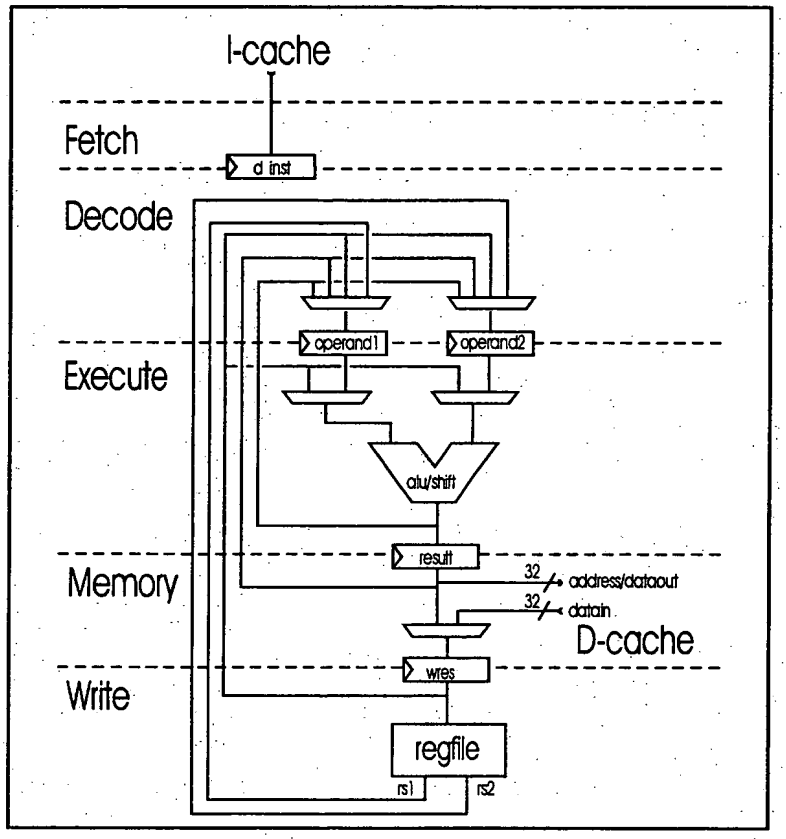


Fig. 50: LEON Pipelined Datapath Structure

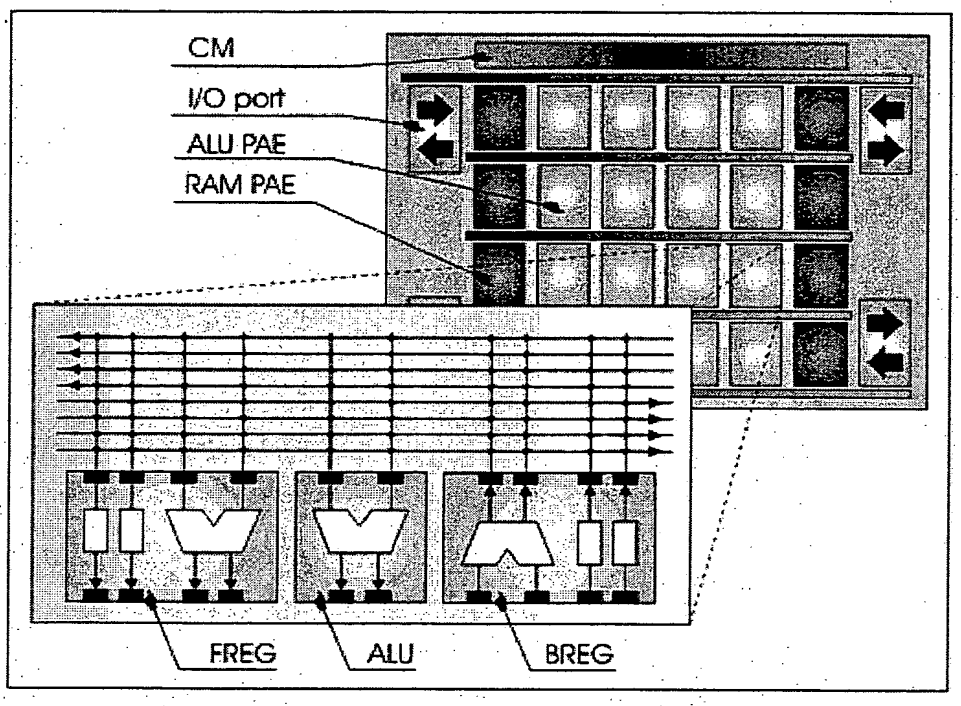


Fig. 51: Structure of an XPP device

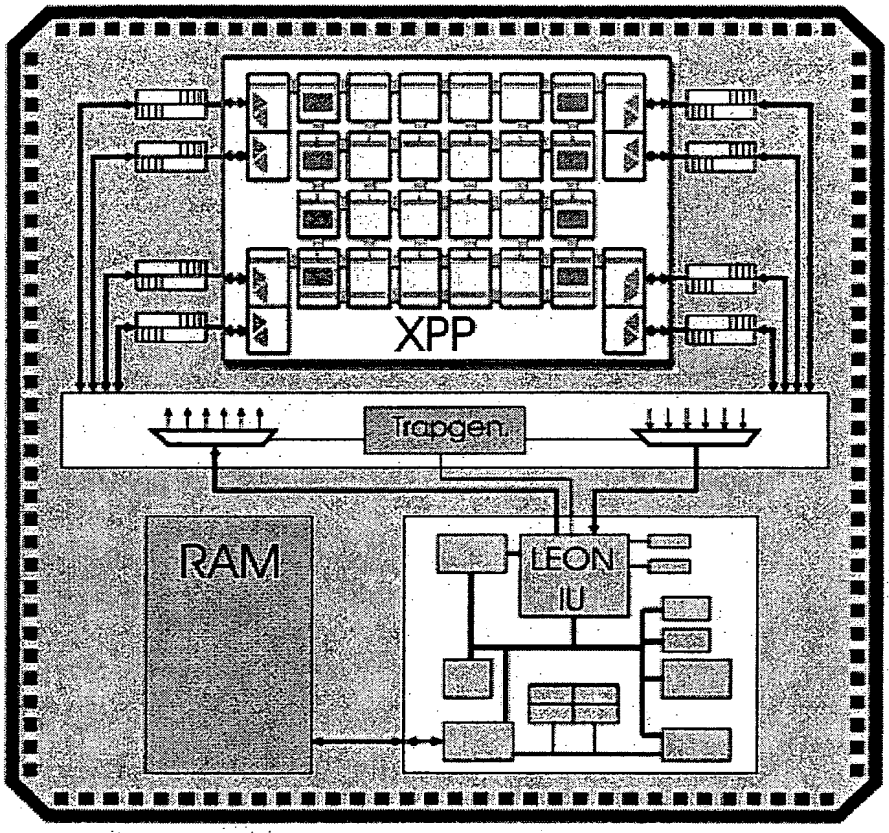


Fig. 52: Extended Datapath Overview

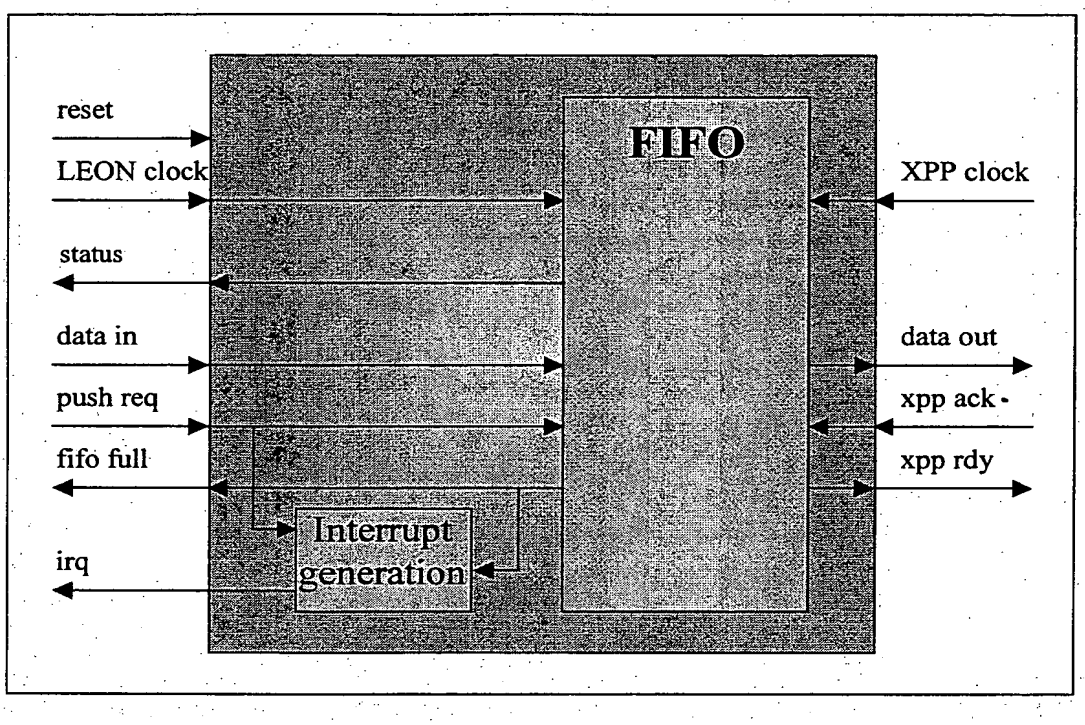


Fig. 53: LEON-to-XPP dual-clock FIFO

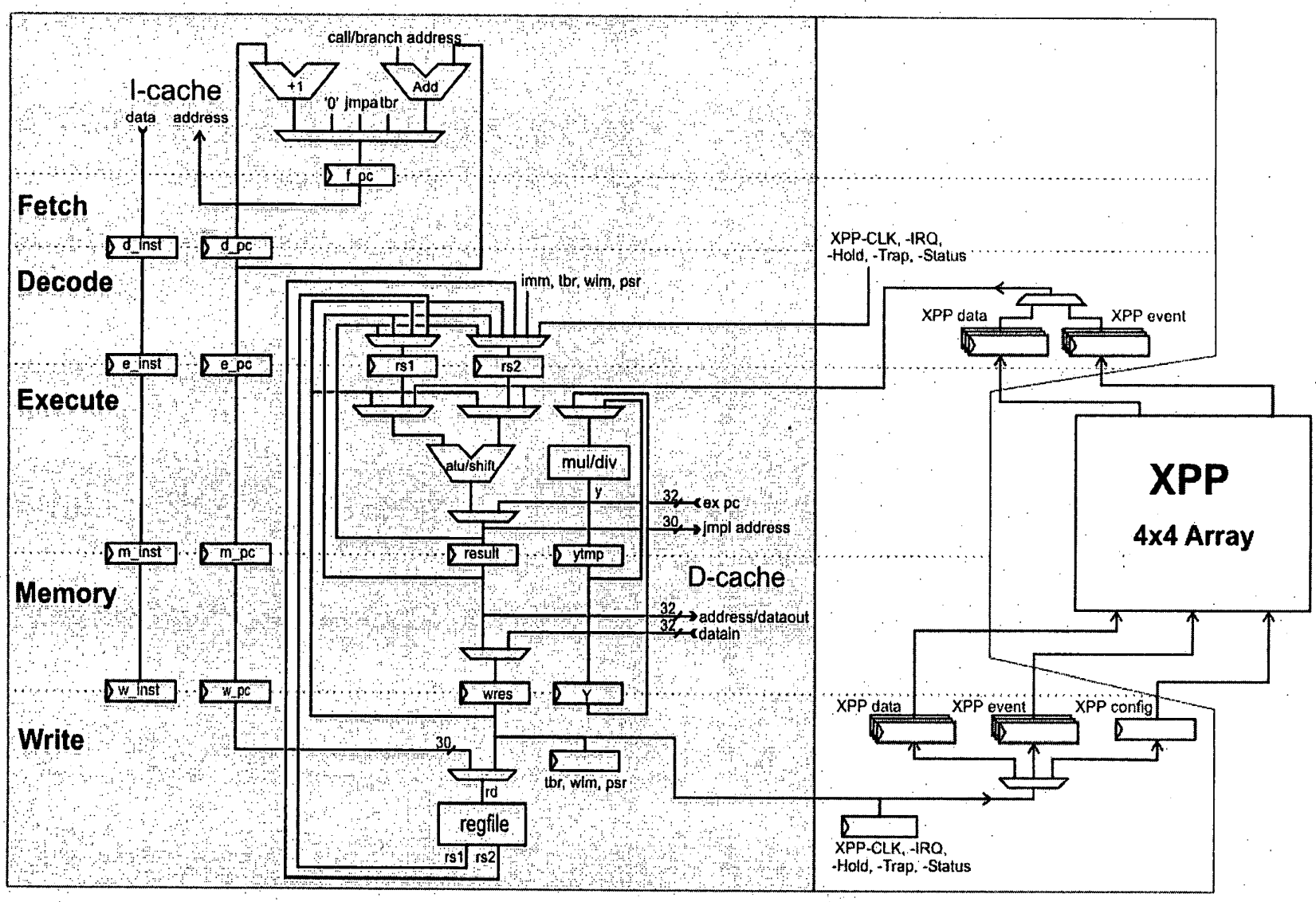


Fig. 54: Extended LEON Instruction Pipeline

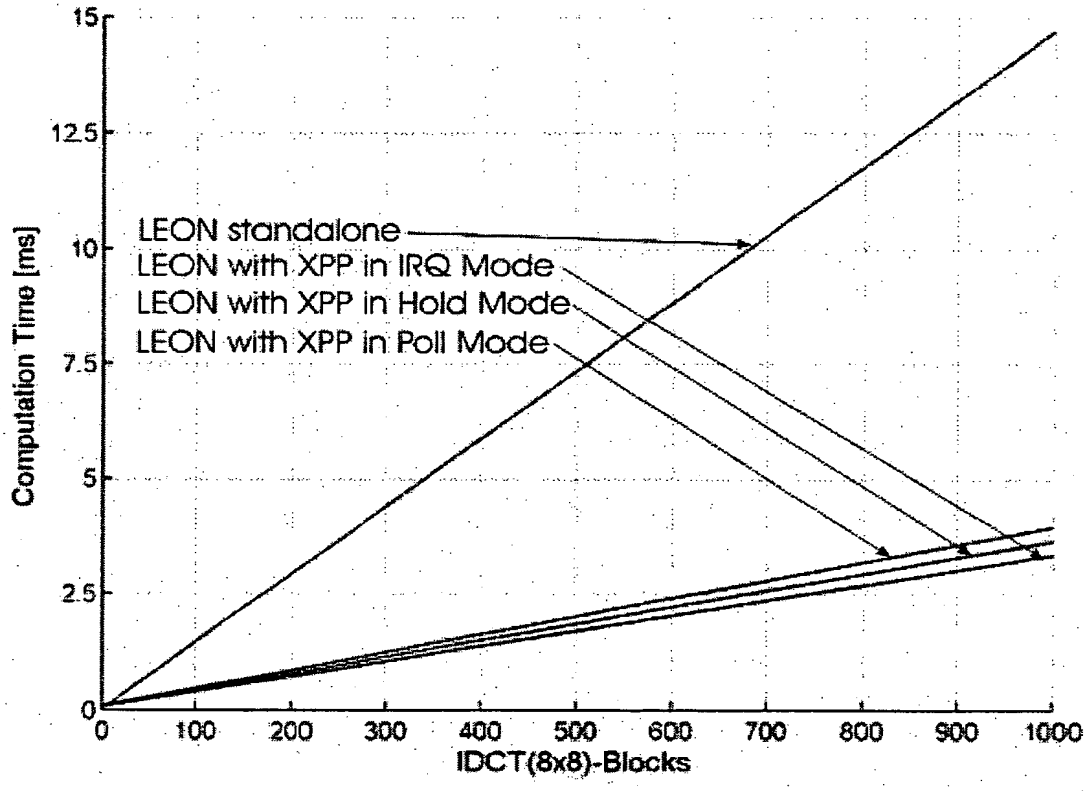


Fig. 55: Computation time of IDCT (8x8)

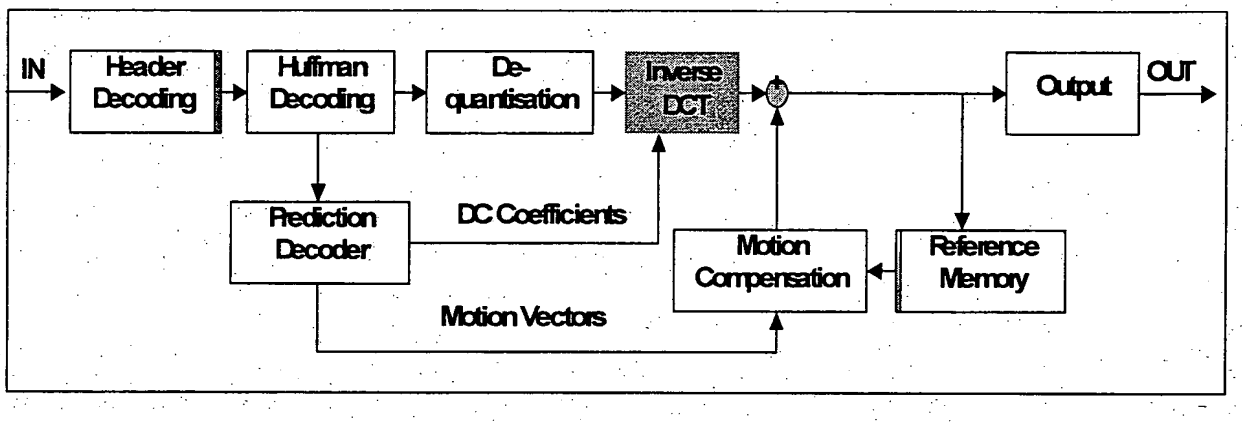


Fig. 56: MPEG-4 Decoder Blockdiagram

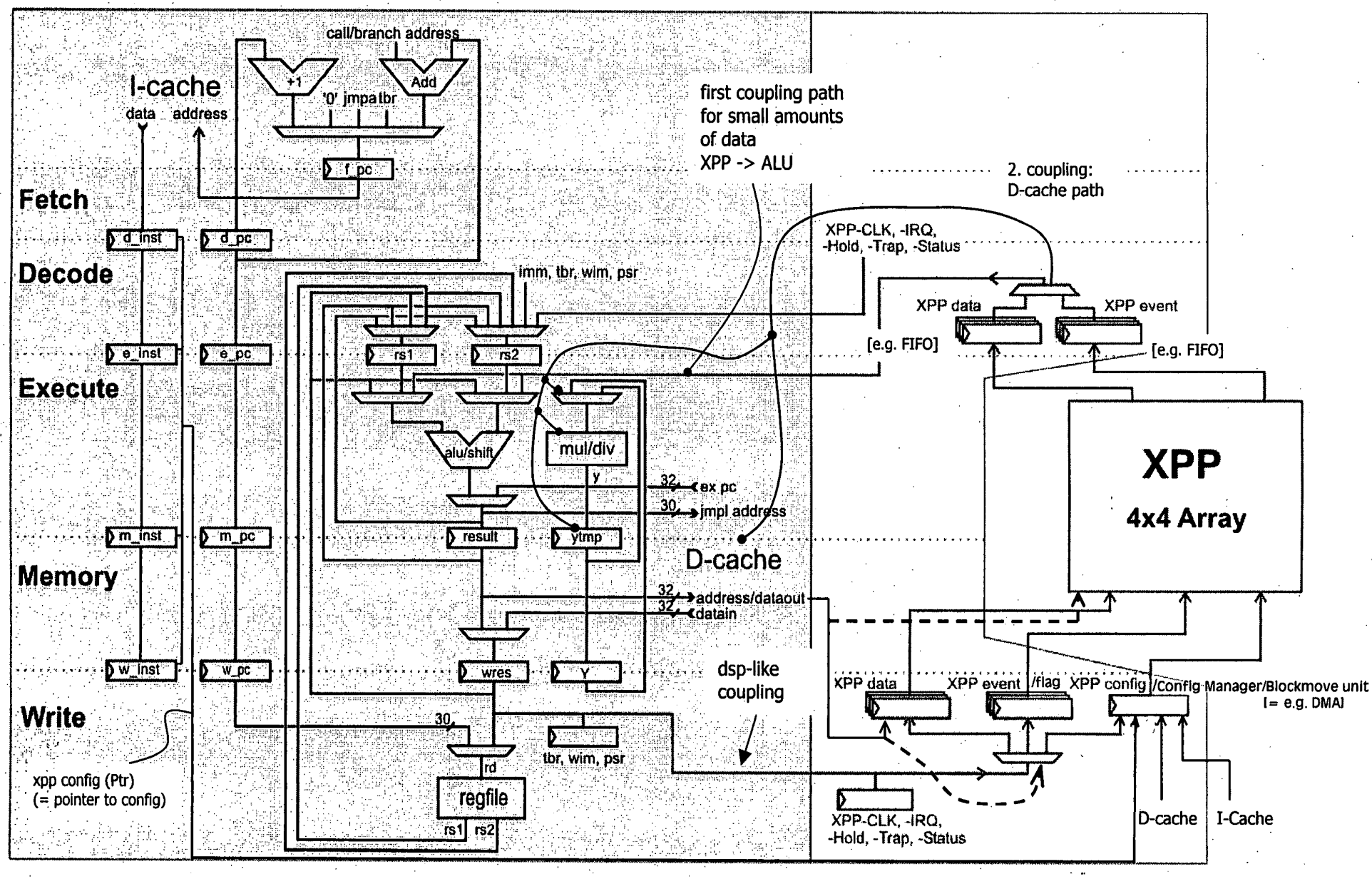


Fig. 57: Extended LEON Instruction Pipeline II

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.